CROSS-LAYER METHODS FOR ENERGY-EFFICIENT INFERENCE USING IN-MEMORY ARCHITECTURES

BY

SUJAN KUMAR GONUGONDLA

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2020

Urbana, Illinois

Doctoral Committee:

Professor Naresh R. Shanbhag, Chair
Professor Pavan Kumar Hanumolu
Assistant Professor Alexander Schwing
Dr. Kailash Gopalakrishnan, IBM

ABSTRACT

In the near future, we will be surrounded by intelligent devices that transform the way we interact with the world. These devices need to acquire and process data to derive actions and interpretations in order to automate/monitor many tasks without human intervention. Such tasks require the implementation of complex machine learning algorithms on these devices. Deep neural networks (DNNs) have evolved into the state-of-the-art approach for machine learning tasks. However, realizing computationally intensive machine learning (ML) algorithms such as DNNs under stringent constraints on energy, latency, and form-factor is a formidable challenge.

In conventional von Neumann architectures, the energy and latency cost of realizing ML algorithms is dominated by memory accesses. To address this issue, the deep in-memory architecture (DIMA) was proposed, which embeds mixed-signal computation as an integral part of the memory read cycle. Deep in-memory architectures have shown up to 100× gains in energy-delay product (EDP) over conventional digital von Neumann architectures. However, the use of mixed-signal computation makes in-memory architectures susceptible to variations and other circuit non-idealities. Therefore in-memory architectures, when implementing ML tasks, exhibit a fundamental trade-off between system-level energy, latency, and accuracy.

Our research focuses on developing cross-layer methods to optimize the system-level energy-latency-accuracy of in-memory architectures for ML applications. First, an automated quantization framework is presented to minimize the precision requirements of DNNs. This framework allocates precision at kernel-level granularity via an iterative greedy process and demonstrates up to 1.2×-1.3× lower precision requirements compared to the state-of-the-art methods on compact networks such as MobileNet-V1.

Next, a compositional framework is proposed that can be used to relate the energy consumption and SNR of in-memory architectures to the various

circuit, architectural, and algorithmic parameters. Analysis using this framework will allow us to design in-memory architecture to meet the application-level precision requirements.

The energy efficiency of DIMA can also be enhanced by the use of compensation techniques to enable a low-SNR operation without any loss in system-level accuracy. The use of stochastic gradient descent (SGD) based on-chip learning to compensate for the impact of chip-specific process variations is studied. The benefits of on-chip learning are demonstrated on a 65 nm prototype integrated circuit (IC) that shows a 2.4× reduction in energy over DIMA operating with off-chip trained weights. When compared to conventional digital architectures, this IC demonstrates up to 100× improvement in energy-delay product (EDP).

In-memory architectures using beyond-CMOS technologies such as STT-MRAM and ReRAM crossbars have become popular due to their advantages in terms of density and scalability. However, such resistive crossbars suffer from inaccurate writes due to device variability and cycle-to-cycle (CTC) variations. We present the Single-Write In-memory Program-vErify (SWIPE) method to achieve high-accuracy writes for crossbar-based in-memory architectures at 5×-to-10× lower cost than standard program-verify methods. SWIPE leverages the bit-sliced attribute of crossbar-based in-memory architectures and the statistics of conductance variations to compensate for device non-idealities.

Extending in-memory computing to storage-class technologies such as NAND flash can be challenging due to stringent density constraints, large capacitances, and low mobility transistors. DIMA for NAND flash memories is introduced, where 8×-to-23× reduction in energy and 9×-to-15× improvement in throughput over the conventional NAND flash systems are achieved. We demonstrate that cross-layer methods are effective in enhancing the system energy, latency, and accuracy of ML systems realized via in-memory architectures.

To my late grandfather Sri Gonugondla Bandeppa.

ACKNOWLEDGMENTS

The last six years have been a long, uncertain, exciting, and fulfilling journey. As I finally submit my dissertation, there are numerous people I would like to thank. I am incredibly grateful to my advisor Prof. Naresh Shanbhag, who guided and motivated me throughout this journey. Prof. Shanbhag's encouragement in improving my skills and pushing the limits of my capabilities has made me a more confident and better researcher. Working with Prof. Shanbhag enabled me to explore and build expertise in a wide variety of topics, often outside my comfort zone, from circuit design to machine learning.

Working with Prof. Shanbhag's group has led me to develop and forester collaborations with other energetic and motivated group members, Mingu Kang, Yongjune Kim, Ameya Patil, Charbel Sakr, Hassan Dbouk, and Sungmin Lim. Each and every one of them brought their unique perspective to the problems I was working on that made my research broad and comprehensive. In particular, Mingu Kang has worked closely with me on the majority of topics in this dissertation and has been a great teacher and motivator. I have also had great opportunities to work with other leading researchers from industry and academia during my Ph.D. Working with Prakalp Srivastava, Prof. Vikram Adve, and Prof. Nam Sung Kim helped me appreciate the research in architecture and compilers. Working with Mark Helm and Sean Eilert from Micron helped me understand the technological challenges in flash memories, which was key in developing in-memory computing for them. I want to extend a sincere thanks to Prof. Pavan K. Hanumolu, Prof. Alexander Schwing, and Dr. Kailash Gopalakrishnan for serving on my doctoral committee and for providing their insightful comments. I want to extend a special thanks to Prof. Pavan Hanumolu for fruitful interactions and feedback on my research. His open and honest opinions pushed us to improve our designs and our research.

I want to thank Mahesh Mehendale and Hetul Sanghvi from Texas Instruments for providing me with the internship opportunity. The internship at Texas Instruments educated me on the challenges I would face in translating my research into the products one could use. Broad and enthusiastic discussions with the team at Texas Instruments encouraged me to think outside the box in tackling the problems in my research.

I could not have asked for a better place than the Coordinated Science Laboratory (CSL) in which to conduct research, thanks to the brilliant, driven, and likeminded individuals I get to interact with every day. I want to thank every group member I had the pleasure to interact with over the course of my Ph.D., notably Sai Zhang, Pourya Assem, Ihab Nahlus, Yingyan Lin, Mingu Kang, Ameya Patil, Charbel Sakr, Yongjune Kim, Sungmin Lim, Hassan Dbouk, Saion Roy, Haocheng Hua, Kuk Hwan Kim, and Rex Geng. They, in many ways, helped shape my research through their suggestions and opinions in meetings and in the office. Numerous open-ended discussions in the group often led to significant breakthroughs that shaped this dissertation.

I am grateful for the University of Illinois and the Department of Electrical and Computer Engineering for providing me the opportunity to pursue my Ph.D. The courses at UIUC with world-class faculty in a wide variety of topics helped me broaden my skills. My stay at the UIUC was made comfortable due to many friendships and relationships I fostered here, especially with Ravi Kiran Raman, Srivathsan Balakrishnan, Rahul Swamy, Meghana Bande, Ameya Patil, and Srilakshmi Pattabiraman. In particular, I am grateful to my wife Meghana Bande, for her love, and for lending a shoulder to lean on. Finally, and most importantly, I would like to thank my parents Surya Prakash and Sunitha Gonugondla, and my sister Sushmitha Gonugondla for their never-ending encouragement and support. I am grateful for all the sacrifices made by my parents, which have led me to where I am today.

I am grateful for the financial support from Systems On Nanoscale Information fabriCs (SONIC) Center and Center for Brain Inspired Computing (C-BRIC) sponsored by the Semiconductor Research Corporation (SRC) and the Defense Advanced Research Projects Agency (DARPA).

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Applications of machine learning algorithms in a variety of fields are being explored today. Machine learning has enabled computers to perform better than their human counterparts in numerous tasks such as image recognition and image segmentation, as well as in complex games such as GO [1]. On the other hand, advances in wireless technology coupled with continuous CMOS scaling over the past decade have enabled a plethora of portable interconnected devices forming the "Internet-of-Things" (IoT).

The deployment of machine learning algorithms on these *Edge* devices will have a broad social and economic impact, e.g., in the areas of public health [2], public policy [3], combating climate change [4, 5], and others (see Fig. 1.1). For many of these applications, Edge devices need to acquire and process data to derive actions and interpretations in real-time. These requirements of portability and real-time response impose constraints on resources such as latency, throughput, and energy of these devices. Furthermore, the ever-increasing complexity of machine learning algorithms adds to these constraints. Realizing machine learning algorithms on computational platforms such as CPUs and GPUs incurs a huge cost in terms of energy and latency, thereby inhibiting their deployment for real-time inference on Edge platforms. These energy and latency costs are dominated by those of memory accesses, e.g., it is well-established that accessing a byte of data from a 32 kB SRAM consumes roughly 10×-to-100× more energy than processing that data in a multiply-accumulate (MAC) unit [6]. This discrepancy in energy and latency costs between memory access and compute increases to 500× for DRAM [6] and 1000× for flash [7].

Memories are a critical component of any modern computing system, allowing it to store data needed for processing. Today's CPUs and GPUs employ a memory hierarchy, designed to maximize the density while minimizing memory access latency and energy. Here, embedded memories that

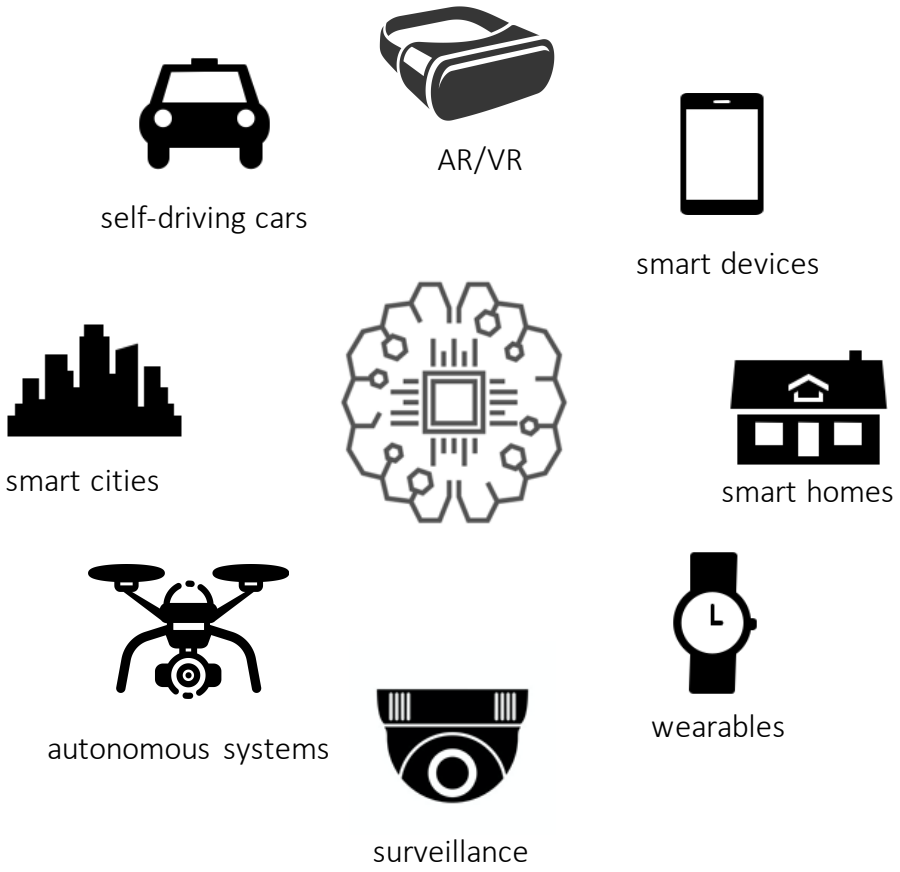


Figure 1.1: Resource constrained applications/platforms that need energy-efficient machine learning acceleration.

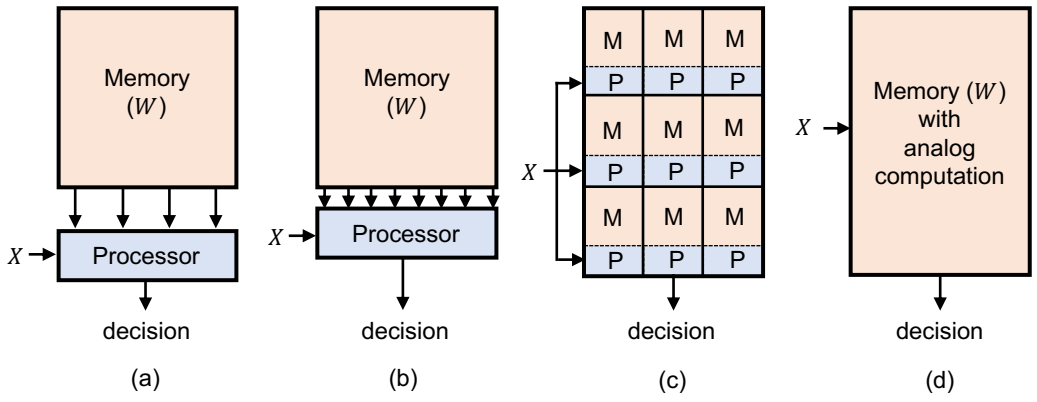


Figure 1.2: Inference architectures: (a) the digital (von Neumann) architecture, (b) the near-memory architecture, (c) the logic-in-memory (LIM) architecture, and (d) the *deep* in-memory architecture (DIMA), where X is an input pattern and W is a stored weight parameter.

form the L1, L2, and L3 caches are the closest to processors and have low access energy, latency, and density. On the other hand, storage class memories such as flash are farthest from the processor and have high density, access energy, and latency. To maximize performance, frequently accessed data is stored in embedded memories. In contrast, less frequently used data is stored farther from the processor in DRAM or flash. A number of compute architectures for machine learning (ML) applications are being explored, as shown in Fig. 1.2. These architectures can be clustered into: (a) digital (von Neumann) architectures, (b) near-memory architectures, (c) logic-in-memory (LIM) architectures, and (d) deep in-memory architectures (DIMAs).

Digital architectures for ML [8–13] (Fig. 1.2(a)) strive to reduce the energy and latency of data accesses via techniques that minimize the use of off-chip memories such as data reuse, data compression, and efficient data-flows. Therefore, these architectures rely on frequent access to embedded memories such as on-chip SRAMs. In fact, Eyeriss [8] and DiaNaNao [13] show that on-chip memory accesses dominate the overall energy consumption of inference.

A variety of compute-in-memory architectures have been proposed to address the costs of on-chip memory access, including near-memory architectures, LIM architectures, and DIMA. Near-memory architectures (Fig. 1.2(b)) distribute computing around the memory banks. Some near-memory architectures embed bit-wise digital operations in the periphery of the memory bitcell array (BCA). For example, [14–16] present SRAM BCAs that implement AND, OR, and XOR operations between two bit-vectors in the sense amplification circuitry. Alternatively, some near-memory architectures use unique 3D technologies that physically stack memory over CMOS logic, either in a monolithic technology [17] or using specialized packaging techniques [18]. Near-memory architectures demonstrate savings on both computational and memory access energy. However, similar to digital architectures, near-memory architectures preserve the intrinsic separation between the memory and the processor.

LIM architectures (Fig. 1.2(c)) use specialized memory cells with digital computing capabilities to realize useful functions. SRAM-based LIM architectures such as [19, 20] implement content-addressable-memory (CAM) for search operations. Others [21–24] employ beyond-CMOS technologies such as RRAM and MRAM to implement CAM. LIM architectures such as [25] embed non-volatile STT-MRAM devices into synthesized CMOS logic for

computer vision applications. By enabling computations in the memory cells, LIM architectures improve the energy and throughput of data access. However, by tightly integrating logic into the memory bitcell, LIM architectures are unable to use memory design *push-rules*, thereby incurring an area overhead of up to 5× [19].

In contrast, DIMA (Fig. 1.2(d)) [26–48] embeds mixed-signal computations with minimal or no alterations to the BCA. DIMA executes vector operations such as dot products as an intrinsic part of the read cycle, thereby avoiding the need to access raw data. Thus, along with the traditional memory operations such as read and write, DIMA allows the users to fetch functions of the data stored. DIMA IC prototypes [28, 30, 49] demonstrate more than 100× reduction in energy-delay product (EDP) compared to equivalent von Neumann digital architectures. Due to the mixed-signal nature of the computations, DIMA exhibits a fundamental trade-off between the EDP and the precision of computations. This trade-off arises due to constraints on the maximum bit-line (BL) voltage swing and the presence of process variations, specifically spatial variations in the transistor threshold voltage V_{t}. Unlike digital architectures, DIMA needs to contend with both quantization noise as well as analog non-idealities caused by process, temperature, and voltage (PVT) variations. This distinction between DIMA and digital architectures raises the following questions:

> *What are the fundamental limits on the energy, latency, and accuracy of DIMA? How do we approach these limits?*

This dissertation attempts to answer these questions. However, doing so is made challenging due to the rich design space occupied by DIMA encompassing a vast diversity of memory devices available, bitcell circuit topologies, architectural options, and algorithmic workloads. While the system-level metrics such as energy, latency, and accuracy are of interest, these metrics depend on the entire compute stack encompassing devices, circuits, architectures, and algorithms. Therefore, this dissertation adopts a cross-layer approach where the proposed solutions cut across algorithms, architectures, circuits, and devices.

At the algorithm-level, we present a methodology that allocates precision at the kernel-level granularity in order to reduce DNN precision requirements. Next, we propose a *compositional framework* that enables us to study

and classify a variety of in-memory architectures and identify the fundamental variables that drive the design choices. The compositional framework also allows us to relate device, circuit, and architectural design variables to system-level metrics such as energy, accuracy, and latency.

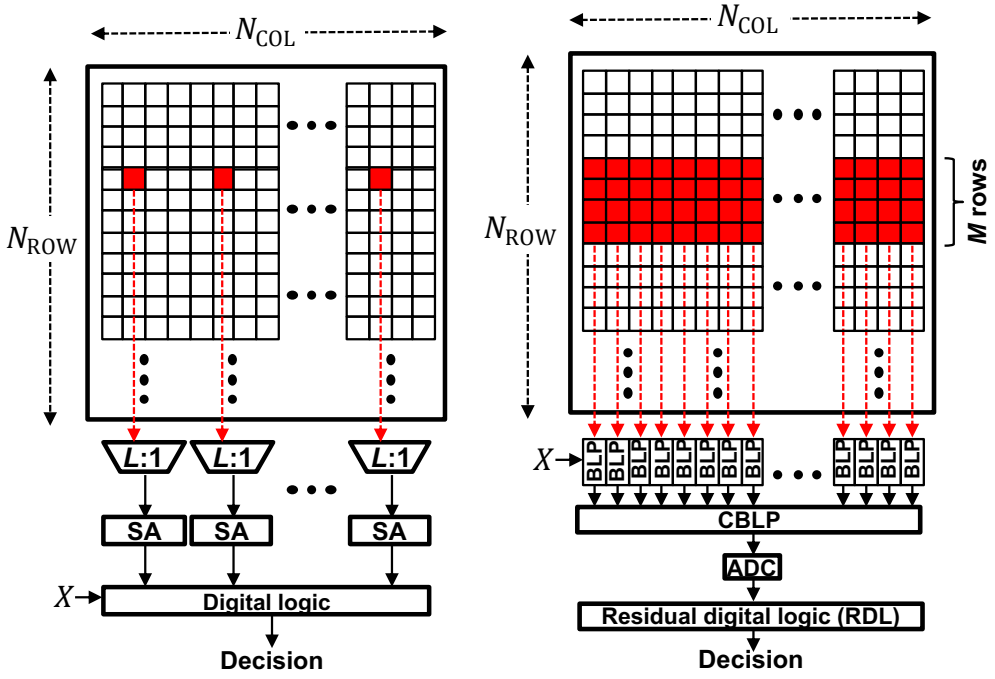
To understand the design trade-offs that impact the energy, latency, and accuracy of DIMA, we adopt a communication-inspired view of DIMA. Here DIMA is seen as a *noisy communication channel* where the energy-efficiency is inherently tied to the accuracy. Using the compositional framework, we derive analytical expressions for energy and SNR as a function of device, circuit, and architectural variables. This analysis enables us to provide design guidelines to develop DIMA that meets the required application-level accuracy and energy metrics.

Akin to error-correction coding, the use of a communication-inspired view allows us to explore cross-layer data-encoding techniques to push the limits of energy and accuracy of DIMA such as the use of on-chip training to adapt and track variations in PVT, and exploiting data statistics to realize a robust in-memory classifier.

Extending in-memory computing to beyond-CMOS memory technologies such as NAND flash, ReRAM, and STT-MRAM is attractive due to the high-density and scalability of these technologies. Despite their benefits, these technologies are highly susceptible to device variations and write noise. We employ a cross-layered approach to develop system and circuit-level techniques that overcome device limitations. A data-encoding procedure to overcome the impact of process variations and write noise in non-volatile memory (NVM) based resistive crossbar architectures is presented. Next, an in-memory architecture for NAND flash memories that preserves the memory array and therefore its density benefits is presented.

1.1 Background and Related Works

This section provides the necessary background on in-memory architectures and describes the prior works that exploit the energy vs. accuracy trade-off in DIMA.



(a) (b)

Figure 1.3: Comparison of: (a) the digital architecture with a $L:1$ column mux and sense amplifiers (SAs) and (b) DIMA [28].

1.1.1 Deep In-Memory Architectures (DIMA)

We define an *idealized* DIMA as one that satisfies the following attributes:

1. **use of a standard BCA:** preserves memory density by using a standard memory bitcell architecture.

2. **row parallelism:** activates multiple rows in the BCA.

3. **bitline computations:** realizes useful analog computations on the bitlines (BLs).

4. **delayed decision:** implements analog computations on the BL outputs to delay hard-decisions (i.e., digitization, comparisons, sense amplifications). This requires the design of column-pitch-matched analog circuits that operate in massively parallel fashion.

In practice, most existing in-memory architectures [26–48] strive to satisfy these attributes to various degrees but none are able to do so fully. Difficulties

in satisfying these attributes arise due to the high compute SNR requirements often imposed by designers and sometimes by applications, stringent density constraints, device variability, technology limitations, and others.

The first DIMA proposed in 2014 at the University of Illinois by Kang *et al.* [26–28, 50, 51] has four sequentially executed processing stages: 1) *functional read* (FR): reads multiple (B) rows of the BCA per access via pulse width and amplitude modulated (PWAM) word-line (WL) access pulses to generate a BL discharge ΔV_{BL} which is proportional to a linearly weighted sum of bits stored in a column-major format (B-way row parallelism and BL computation); 2) *BL processing* (BLP): processes BL discharge ΔV_{BL} voltages in a massively parallel (single-instruction multiple data (SIMD)) fashion via column pitch-matched mixed-signal circuits to execute scalar arithmetic operations such as addition, subtraction, multiplication, and comparison (delayed decision); 3) *cross BL processing* (CBLP): aggregates the BLP outputs via charge-sharing to obtain a scalar analog output (delayed decision); and 4) *residual digital logic* (RDL): converts the CBLP's analog output into a digital word and then executes any residual functions digitally to generate the final inference.

In contrast to the conventional digital architecture (Fig 1.3(a)), DIMA (Fig 1.3(b)) reads B rows of the BCA per access and avoids the need for a $L:1$ (typically $4:1$ to $16:1$) column multiplexer prior to sense amplification, where L is the number of BLs sharing a single sense amplifier. Therefore, DIMA processes the same number of bits with fewer ($1/LB$, which is typically 1/16 to 1/64) read cycles leading to large energy and latency gains. Silicon prototypes of DIMA [28,30,52] show up to 9.7× reduction in energy and 5.3× improvement [28] in throughput over fixed-function digital implementations.

DIMA, as proposed in [26–28], preserves: a) memory density by employing conventional 6T bit-cell memory architecture, and b) generality by enabling multi-bit vector operations. Subsequently, other variants of DIMA were proposed that made alternate design choices to either enhance scalability and/or robustness by: 1) restricting one or both operands to binary (1-b) precision [32, 43], 2) using larger (8T or 10T) bit-cells [29], 3) using specialized bit-cells that enable computations [31], 4) partitioning the array into sub-banks [29], and 5) using a separate right and left word-lines [32].

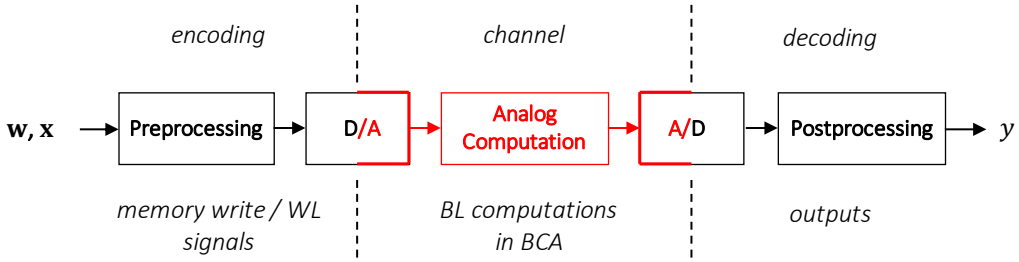


Figure 1.4: A communication inspired-view of DIMA.

1.1.2 A Communication-Inspired View of DIMA

Though DIMA is inherently a *digital-in digital-out* architecture, the computations are performed in mixed-signal domain (see Fig 1.4). In contrast to digital architectures, the mixed-signal nature of DIMA introduces additional noise sources that determine the accuracy of those computations. The impact of these noise sources on the system-level accuracy depends on various design choices across the compute stack from devices, circuit typologies, architectural choices, and application workloads. These noise sources ultimately determine the fundamental limits on achievable energy-efficiency and latency of inferences generated by DIMA. Thus DIMA offers numerous design-variables that trade accuracy for energy-efficiency. For example, measurements of DIMA in Fig. 1.5 show the trade-off between the dot product computational energy and the impact of variations on the functional read operations. The design variable that controls this trade-off is the BL discharge voltage $\Delta V_{\mathrm{BL,max}}$. In order to understand inherent energy vs. accuracy trade-offs in DIMA, we adopt a communication-inspired view where data procession in DIMA is seen as data transmission across a noisy channel.

Many recent works recognize and exploit the intrinsic energy-vs-accuracy trade-off in DIMA. For example, PROMISE [54] appropriately chooses $V_{\mathrm{BL,max}}$ based on the precision requirements at the application level. An alternative way to enhance energy-savings in DIMA is to employ algorithmic redundancy so that the final accuracy metrics are robust to noise and circuit non-idealities. For example, [30] implements a series of weak and noisy classifiers on DIMA and then aggregates their results using AdaBoost to achieve a robust high-accuracy classifier. Similarly, [55] employs the random forest (RF) algorithm, where classification results made by multiple binary decision trees on DIMA are aggregated to achieve a robust classification. Alternatively, [56]

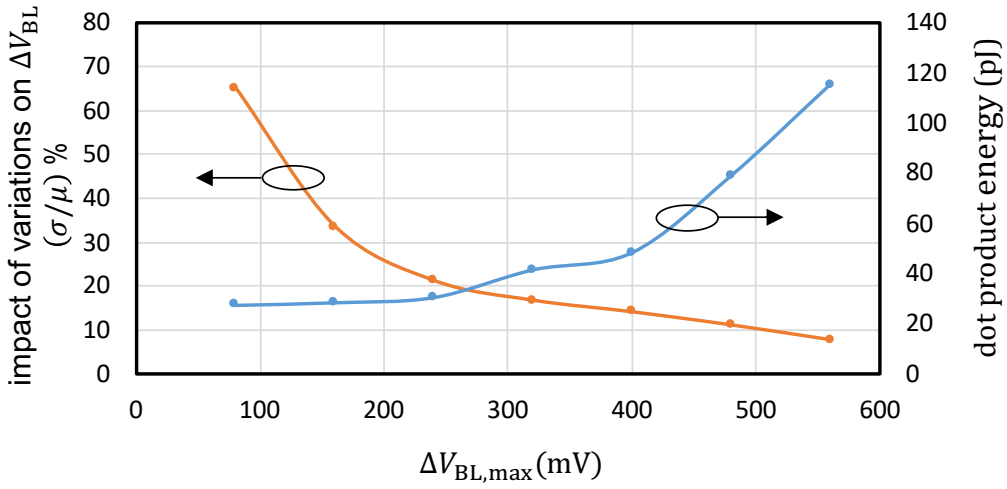


Figure 1.5: Measurements showing the impact of process variations and the total dot product computation energy in DIMA [53].

employs recurrent attention models (RAM) that progressively improve the classification accuracy by processing the input data over multiple cycles. Statistical error compensation techniques [57, 58] such as algorithmic noise tolerance can also be used with to overcome the impact of process variations. While the algorithmic redundancy is undoubtedly useful in reducing the SNR requirements of DIMA operations, it is not clear if the resulting energy savings justify the use of additional computations. A clear understanding of energy-vs-SNR trade-offs in DIMA is required to resolve this conundrum. To do so, we employ a compositional framework for in-memory architectures to analyze DIMA and analytically characterize its energy-vs-SNR trade-offs.

1.2 Dissertation Contributions and Organization

This dissertation attempts to understand and approach the limits of energy, latency, and accuracy of DIMA. The challenge in achieving this goal is the vast design space of DIMA that spans architectures, circuits, process technologies, and applications. This dissertation adopts a cross-layered approach to address this challenge by: (1) minimizing the precision requirements of the algorithms via a granular precision assignment for DNNs, (2) developing a compositional framework to analyze and explain DIMA, (3) characterizing the SNR and energy of in-memory computations using the compositional

framework, (4) employing on-chip learning to overcoming the impact of process variations and thereby push the limits of energy efficiency, (5) proposing data-encoding techniques to overcome the impact of write-noise in crossbar memories, and (6) extending DIMA to high-density memories such as NAND flash. The contributions and organization of the dissertation are summarized as follows:

Chapter 2 introduces the iterative mixed-precision quantization (IMPQ) methodology where precision is allocated at the kernel-wise granularity in DNNs. A granular precision assignment is challenging due to the search space that grows exponentially with the number of kernels. IMPQ employs an iterative process where precision is reduced in the least sensitive kernels followed by retraining to regain the accuracy loss. Compared to other state-of-the-art granular precision assignment methods, IMPQ reduces the storage requirements by 1.2×-1.3× for compact networks such as MobileNet-V1 using weight-only quantization. Results for both weight and activation quantization are also presented. Our experiments show that granular precision assignment provides better compression than allocating precision uniformly across all layers. We interpret the implication of the IMPQ and its impact on neural network architectures.

Chapter 3 proposes a compositional framework to explain and study DIMA. In particular the construction of DIMA is framed as a combination of in-memory compute models, and multi-bit dot product mapping techniques. Specifically, this chapter describes the three *in-memory compute models* (charge accumulation (QA), current accumulation (QA), and charge sharing (QS)), and the associated noise and energy models. This framework enables us to explain the functionality, trade-offs, and limits on energy-efficiency and accuracy of a large class of recently published in-memory IC prototypes.

Chapter 4 presents a methodology to analyze the compute SNR of DIMA. It leverages the compositional framework, compute models, and associated noise models from Chapter 3 to analytical expressions for the compute SNR of DIMA dot products. This analysis exposes the relationship between SNR, energy, input precision, weight precision, output precision, dot product length, and other design variables such as noise variances. The analysis presented in this chapter enables one to make appropriate design choices to meet the system-level accuracy and energy requirements. Based on the analysis, this

chapter presents a set of guidelines for design efficient DIMA.

Chapter 5 studies the use of an on-chip SGD-based trainer to adapt and track variations in PVT and data statistics. This is demonstrated via a DIMA-based 65 nm CMOS IC implementing vector machine (SVM) classifier. SGD-based algorithms are commonly employed to train many ML systems, including deep neural networks (DNNs). Measurement results show that on-chip learned weights enable accurate inference in the presence of scaled BL voltage swing, thereby leading to a 2.4× greater energy savings over an off-chip trained DIMA implementation. Compared to a conventional fixed-function digital architecture with identical SRAM array, the prototype IC achieves up to 21× lower energy and 4.7× smaller delay leading to a 100× reduction in the EDP.

Chapter 6 proposes the Single-Write In-memory Program-vErify (SWIPE) method to achieve high-accuracy writes for crossbar-based DIMA at 5×-to-10× lower cost than standard program-verify methods. SWIPE leverages the bit-sliced attribute of crossbar-based DIMA and the statistics of conductance variations to compensate for device non-idealities. Using SWIPE to write into ReRAM crossbar allows for a 2× (CIFAR-10) and 3× (MNIST) increase in storage density with $< 1\%$ loss in DNN accuracy. In particular, SWIPE compensates for 4.8×-to-7.7× higher conductance variations. SWIPE augmented with noise injection based training methods achieves further enhancements in robustness.

Chapter 7 presents DIMA for NAND flash memories (DIMA-F) to accelerate computations in big data applications. The energy and throughput costs of NAND flash memories are severely bottle-necked by its off-chip I/O costs. By introducing computational capabilities inside the NAND flash arrays, we reduce the need for off-chip data transfers. DIMA-F achieves up to 8×-to-23× reduction in energy and a 9×-to-15× reduction in throughput resulting in EDP gains up to 345× over the conventional NAND flash architecture.

Chapter 8 summarizes the contributions and future research directions.

In the rest of this dissertation, the terms *in-memory architectures* and *DIMA* are employed interchangeably.

CHAPTER 2

REDUCED COMPLEXITY NEURAL NETWORKS VIA GRANULAR PRECISION ASSIGNMENT

Application-level precision requirements play an important role in determining the energy-efficiency of in-memory architectures. Such architectures use analog mixed-signal techniques, which are known to be energy-efficient for low-precision computations. For example, analytically, it has been shown that the analog computations are more energy-efficient than their digital counterparts for SNR $< 60\,\mathrm{dB}$ (equivalent to 8-b to 9-b fixed-point arithmetic) [59]. Not surprisingly, many recent IC realizations of in-memory architectures implement dot products with input and weight precision in the 4-b to 8-b range [28, 35, 41, 60]. Therefore, techniques that reduce the precision requirements of DNNs will have a significant impact on the applicability of in-memory architectures to ML tasks.

This chapter presents the iterative mixed-precision quantization (IMPQ) methodology to obtain a mixed-precision DNN where precision is assigned kernel-level granularity. Such granular precision assignment is a challenging problem due to the exponentially large search space. IMPQ employs an iterative greedy process to arrive at a good solution. We demonstrate the effectiveness of our approach with a compact network such as MobileNet-V1. The benefits of the granular precision assignment are discussed and demonstrated via experiments.

2.1 Background and Related Works

Reducing the DNN complexity has been an active area of research. Techniques such as network pruning [61] were developed in order to remove redundant weights and computations without compromising on accuracy. However, these techniques result in an irregular network structure that is challenging to implement in regular array-based hardware such as in-memory architectures.

An alternative technique to reduce DNN complexity is to design compact network architectures from scratch, e.g., MobileNets [62], SqueezeNet [63], ShuffleNet [64], and ConDenseNet [65]. These networks employ specialized layers such as *depth-wise separable layers*, *grouped convolutions*, *point-wise convolutions*, and others. These networks are designed to achieve high accuracy with far fewer computations and parameters. For example, the classification accuracy of MobileNet-V1 on ImageNet-1k dataset is close to that of VGG-16 [62], but with 29× fewer operations and 33× fewer parameters.

Recently, training methods were developed that use binary and ternary precision for weights and activations [66, 67]. These techniques applied to large networks such as VGG-16 and ResNet-18 result in minimal accuracy drop from their floating-point counterparts. However, ternarization applied to the weights in the MobileNet-V1 result in a catastrophic accuracy drop. Hence, aggressive quantization on compact networks such as MobileNet-V1 still remains a challenge.

Conventional quantization techniques [68, 69] use the same number of bits across all layers. However, it is observed [70] that the sensitivity of inference accuracy to quantization noise varies across layers, leading to different per-layer precision requirements. Works such as [71, 72] have also observed that the precision requirements vary across different kernels within the same layer. This diversity across layers and kernels can be exploited by in-memory architectures and other variable precision hardware [9, 12, 35]. In spite of its benefits, kernel-level granular precision assignment remains a challenge due to the vast complexity of the search space. For example, N-layered neural network with up to B-b weights and activations has B^{2N} possible layer-wise precision assignments, e.g., there are greater than 4 billion possibilities for VGG-16 if each layer has four possible precision assignments.

We classify the research on developing a quantized neural network in the following categories:

Fixed point network without training: These methods [70, 73, 74] strive to obtain a quantized neural network from a pre-trained floating-point counterpart analytically, i.e., without having to retrain it, thereby avoiding the huge cost of retraining.

One such technique is *noise gain analysis* (NGA) proposed by Sakr *et al.* in [70, 73]. Here, given a floating-point baseline and the precision assignment, [70, 73] provide an analytical expression upper bound on the mismatch

probability, $p_m = P(\hat{Y}_{\text{fx}} \neq \hat{Y}_{\text{fl}})$ which is the probability that the class label predicted by the floating-point baseline ($\hat{Y}_{\text{fl}}$) does not match the class label predicted by the fixed-point network ($\hat{Y}_{\text{fx}}$). This bound on p_m is given by:

$$p_m \leq \sum_l \Delta_{A,l}^2 E_{A,l} + \sum_k \Delta_{W,k}^2 E_{W,k} \tag{2.1}$$

where $\Delta_{A,l}^2 = 2^{-(B_{A,l}-1)}$, and $\Delta_{W,k}^2 = 2^{-(B_{W,k}-1)}$ are the quantization step-size of the set of activations $\mathcal{A}_l$ and the set of weights $\mathcal{W}_k$, respectively. The noise gains $E_{A,l}$ and $E_{W,k}$ quantify the impact of quantization noise of $\mathcal{A}_l$ and $\mathcal{W}_k$ on the mismatch probability, respectively.

Analytical expressions for the noise gains are given by:

$$E_{A,l} = \mathbb{E}\left[\sum_{i \neq \hat{Y}_{\text{fl}}} \frac{\sum_{a \in \mathcal{A}_l} \left|\frac{\partial (Z_i - Z_{\hat{Y}_{\text{fl}}})}{\partial a}\right|^2}{24|Z_i - Z_{\hat{Y}_{\text{fl}}}|^2}\right] \tag{2.2}$$

and

$$E_{W,k} = \mathbb{E}\left[\sum_{i \neq \hat{Y}_{\text{fl}}} \frac{\sum_{w \in \mathcal{W}_k} \left|\frac{\partial (Z_i - Z_{\hat{Y}_{\text{fl}}})}{\partial w}\right|^2}{24|Z_i - Z_{\hat{Y}_{\text{fl}}}|^2}\right] \tag{2.3}$$

where Z_i is the soft output of the DNN that corresponds to the class label i. Empirically, the noise gains are estimated by taking gradients with respect to margins on a subset of the training samples during the standard back-propagation phase of training.

Fixed precision techniques: These methods propose training a DNN to a specific predetermined precision such as a binary or ternary precision. Early works such as [75] focused on techniques to aggressively quantize both weights and activations to fixed valued binary representation (+1/-1). XNOR-Net [76] improved on its accuracy by providing additional degrees of freedom by using scaled binary values to represent the network weights and activation, i.e., $(\alpha, -\alpha)$, where α is a floating-point scaling factor chosen on a per-kernel basis. Ternary quantization techniques such as [66] introduced the 0 state that led to a huge boost in accuracy. Subsequent works such as [67, 77] enhance on the classification accuracy by improving on the training methodology. However, these works limited themselves to a fixed precision assignment (binary/ternary) and do not translate to other precision

assignments.

Network-wise precision assignment: These methods focus on training a DNN with given a precision assignment across the network. Works such as DoReFa-Net [68] and PACT [69] train networks with an arbitrary assignment. DoReFa-Net [68] introduced a floating-point multiplier based on the maximum value of the floating-point weight. This value was assigned on a per-layer basis to weights and activation. PACT [69] extended this technique by parameterizing the multiplier and learning it via back-propagation. Non-uniform network quantization techniques such as LQ-Net [78] and ABC-Net [79] interpret multi-bit weights and activation as a linear combination of binary-valued tensors. These works use the same precision across the network and they need the network to be trained from scratch for each precision assignment. Determining the appropriate precision can itself be a challenge. It is particularly tricky with non-uniform quantization techniques that are computationally intensive to train with.

Granular precision assignment: Granular precision assignment across layers is challenging due to the search space that increases exponentially with the number of layers. Mixed-precision network training techniques [80, 81] improve the accuracy given a precision assignment but they face an exponentially large search space. Numerous techniques to assist the search for a good precision assignment in the search space have been proposed. HAQ [82] uses a reinforcement learning (RL) agent to pick a precision assignment by iteratively evaluating the network on a hardware model. Alternatively, HAWQ [72] uses the second derivative information to find the sensitivity of the layers. AutoQ [71] extends the ideas presented in HAQ to implement kernel-wise precision assignment. HAQ and AutoQ need to train an RL agent which is computationally expensive. Furthermore, HAWQ needs to estimate the Hessian diagonal of weights to determine sensitivity, which is computationally challenging.

In summary, though granular precision assignment is effective in reducing the overall network complexity, it is challenging to find a good solution due to its exponential large search space.

2.2 Proposed Iterative Mixed-Precision Quantization (IMPQ)

IMPQ employs a four-step iterative process: a) evaluate sensitivity, b) pick weight/activation groups, c) reduce precision, and d) fine-tune network.

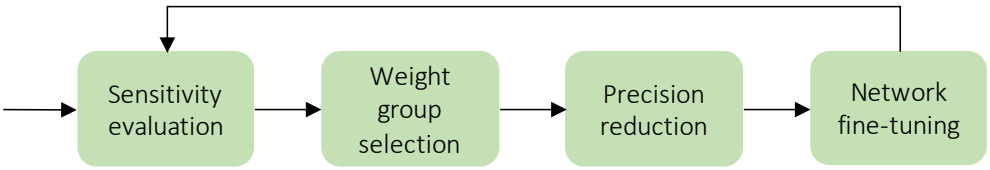


Figure 2.1: The proposed iterative mixed-precision quantization (IMPQ) methodology.

IMPQ (see Fig. 2.1 and Algorithm 1) begins with a pre-trained network where weights and activations quantized to an identical precision across all layers and kernels. The precision of this pre-trained network is chosen to meet the state-of-the-art accuracy of a floating-point network, and therefore it is the maximum allowable precision in the final mixed-precision network. We use the term *weight group* or *activation group* to refer to the granular block having the same precision. Weight groups can be at a kernel-wise or layer-wise granularity, and activation groups are at a layer-wise granularity. Kernel in this work refers to a 3 dimensional filter that is convolved with the input feature maps to obtain an one output feature map.

Sensitivity Evaluation: Sensitivity estimates the impact of perturbations of weights or activations on the classification accuracy. We use the method proposed in [70, 73] to evaluate the sensitivity to quantization noise. However, instead of comparing a fixed-point network to a floating-point network, we are interested in evaluating the impact of reducing precision for a specific weight group or an activation group in an already quantized network. Therefore, we adapt the noise gains (2.2) and (2.3) for a quantized network given the predicted class $\hat{Y}$ as follows:

$$M_{A,l} = 2^{-B_{A,l}} \mathbb{E}\left[\sum_{i \in S} \frac{\sum_{a \in \mathcal{A}_l} \left|\frac{\partial (Z_i - Z_{\hat{Y}})}{\partial a}\right|^2}{24 |Z_i - Z_{\hat{Y}}|^2}\right] \tag{2.4}$$

and

$$M_{W,k} = 2^{-B_{W,k}} \mathbb{E}\left[\sum_{i \in S} \frac{\sum_{w \in \mathcal{W}_k} \left|\frac{\partial (Z_i - Z_{\hat{Y}})}{\partial w}\right|^2}{24|Z_i - Z_{\hat{Y}}|^2}\right] \tag{2.5}$$

where $M_{A,l}$ and $M_{W,k}$ are the sensitivities of activation group $\mathcal{A}_l$ and the weight group $\mathcal{W}_k$, respectively, S is a set of class labels excluding $\hat{Y}$, and $B_{A,l}$ and $B_{W,k}$ are the precisions of l-th activation group and k-th weight group, respectively.

Weight/Activation Groups Selection: In the second step of IMPQ methodology, we use the sensitivity estimated via (2.4) and (2.5) to identify the weight/activation groups that have the least impact on mismatch probability. There are three possible ways to pick these weight/activation groups: a) thresholding the sensitivity, b) thresholding mismatch probability, and c) fixed the number of groups chosen.

Precision Reduction: We reduce the precision of the weight/activation groups that were chosen in the previous step. This may result in an accuracy drop that needs to be recovered.

Network Fine-tuning: Here we train the current network usually for a few epochs on the training data.

Each iteration of IMPQ takes a small step in reducing the overall precision requirements. The network at the end of each iteration is used to determine if the complexity is sufficiently reduced as per the user requirements. The specifics of the experiments and approximations used for a practical implementation of this methodology are presented in the next section.

2.3 Implementation Details

This section lists the implementation challenges and procedures employed for the experiments.

Quantization: IMPQ works for both uniform and non-uniform quantization. Experiments in this chapter use uniform quantization for both weights and activations. For each weight w in k-th weight group, we first clamp the weights between to a range $[d, -d]$ and then quantize it uniformly within this

Algorithm 1 Proposed iterative mixed-precision quantization (IMPQ) methodology

Input: Neural network architecture $f(\mathbf{X}, \mathcal{W})$; floating point reference weights $\mathcal{W}_{k,f} \subset \mathcal{W}_f$; quantized weights $\mathcal{W}_k \subset \mathcal{W}$ are the k-th weight group with precision $B_{w,k}$; $\mathcal{A}_l \subset \mathcal{A}$ are the l-th activation group with precision $B_{a,l}$; training data and labels $\{(\mathbf{X}_j, \mathbf{y}_j)\}$ where j is index of the training batch; number of training batches N_{Batch}; number of groups picked for precision reduction N_{group}; and the number of iterations of IMPQ N_{iter}.

Output: Quantized network weights.

Initialize: $\mathcal{W}_f$,, $B_{W,k} = B_{W,start}$ and $B_{A,l} = B_{A,start}$

function QUANTIZEDTRAIN
    $i \leftarrow i + 1$
    while not converged **do**
        $\mathbf{y}'_i \leftarrow f(\mathbf{X}_i, \mathcal{W})$ ▷ Forward propagation
        Evaluate $\mathcal{L}(\mathcal{W}, \mathbf{y}'_i, \mathbf{y}_i)$ ▷ Loss function
        $\mathcal{W}_f \leftarrow \mathcal{W}_f - \nabla\mathcal{L}(\mathcal{W}, \mathbf{y}'_i, \mathbf{y}_i)$ ▷ Weight update
        $\mathcal{W}_k \leftarrow \text{Quantize}(\mathcal{W}_{k,f}, B_{w,k})$
        $i \leftarrow 0$ if $(i == \text{NBatch})$ $i + 1$ otherwise
    end while
end function
QuantizedTrain() ▷ Train initial network
for $k := 1$ to N_{iter} **do**
    $\mathbf{z} \leftarrow f(\mathbf{X}, \mathcal{W})$ ▷ Forward propagation
    Evaluate sensitivity $M_{W,l}$ using (2.5)
    SI $\leftarrow$ SortIdx($M_{W,l}$) ▷ Sorts weight groups based on $M_{W,l}$
    for $n := 1$ to N_{group} **do**
        $r \leftarrow \text{SI}(n)$
        $B_{W,r} \leftarrow B_{W,r} - 1$ ▷ Reduce precision
    end for
    QuantizedTrain()
end for
Repeat for activations

range. The quantized weight w_q corresponding to a weight w is given by:

$$w_q = \text{round}\left(\frac{2^{B_w-1}\min(\max(w,-d),d)}{d}\right)2^{-B_w+1}d \tag{2.6}$$

The clipping value d is an important parameter and needs to be carefully chosen. Most works, such as [68, 73], choose it based on the maximum value of the weight group. Alternatively, [69] parameterizes d and learns it via back-propagation, and [82] uses KL divergence to choose d. In this work, we choose the clipping value d as a multiple of the second moment of the weight group $\mathcal{W}_k$, as follows

$$d = 4\sqrt{\frac{\sum_{w\in\mathcal{W}_k} w^2}{|\mathcal{W}_k|}} \tag{2.7}$$

Choosing d based on the second moment of the weights minimizes its sensitivity to outliers, thereby ensuring its stability during training.

Activations are unsigned and are quantized between $[0, d]$. Since ReLU activation function followed by a batch-norm layer is used, we choose d as follows:

$$d = \max(\beta_i + 6\gamma_i) \tag{2.8}$$

where β_i and γ_i is the shift and scale batch-norm parameters of the i-th feature map. Assuming the activation follows a Gaussian distribution, this clipping value ensures that less than 0.1% of the activations are clipped.

Approximations to sensitivity: Estimating the sensitivity as per (2.4) and (2.5) requires us to estimate the gradients of the weights with respect to each class probability. This poses a few practical challenges.

Since (2.4) uses a sum of squared gradients rather than a linear sum, neural network training packages such as PyTorch cannot be directly used, and back-propagation cannot be performed for an entire batch in one-shot. Instead, per-sample back-propagation needs to be performed making it challenging to estimate the sensitivity metrics for the entire dataset. Furthermore, the number of back-propagation that need to be performed per sample scales with number of class labels, i.e., if there are M classes in the dataset, the back-propagation needs to be applied $M-1$ times per input sample.

To address this computational challenge, we approximate (2.4) and (2.5) by choosing only a fraction of the training data. Additionally, we consider the gradients with respect to a few classes with the smallest margins, i.e.,

the set S in (2.4) and (2.5) is a subset of the set of class labels.

2.4 Experimental Results

We demonstrate the effectiveness of IMPQ with experiments on the following image classification datasets: 1) CIFAR-10 [83], 2) SVHN, and 3) ImageNet-1k [84]. We choose ResNet-20 [85] and VGG-Small [78] for classifying the CIFAR-10 dataset, and MobileNet-V1 for classifying the ImageNet-1k dataset. For these experiments, we train a floating-point and baseline networks from scratch. We use the same hyper-parameters, such as the learning rate, transformation, and augmentation for all the networks, including the floating-point baseline architecture. We used stochastic gradient descent with weight decay for all the architectures.

2.4.1 Weight-only Quantization

We first demonstrate the effectiveness of IMPQ with weight-only quantization. Unlike other techniques such as PACT [69], HAQ [82], and LQ-Nets [78], we quantize all the layers, including the first and the last fully connected layers. We evaluate our results using the *effective precision* metric defined as:

$$B_{w,\text{eff}} = \frac{\sum_k B_{w,k}|\mathcal{W}_k|}{\sum_k |\mathcal{W}_k|} \tag{2.9}$$

Impact of Network Complexity: Figures 2.2 and 2.3 show how the accuracy and the effective precision change with multiple iterations of IMPQ on ResNet-20 and VGG-Small respectively. The precision of the initial pretrained network on ResNet-20 and VGG-Small is 4-b and 2-b, respectively.

At iso-accuracy, IMPQ reduces the effective precision by >42% with respect to LQ-Net on ResNet-20 (see Table 2.1). In contrast, over-parameterized networks such as VGG-Small (17× more parameters than the more compact ResNet-20) can operate with very few bits and may not require mixed-precision techniques (see Table 2.2). Similar results were observed on the SVHN dataset using the VGG-Small network architecture (see Table 2.3).

Similarly, unlike large networks such as VGG-16 that can operate with ternary precision with minimal loss in accuracy, MobileNet-V1 is sensitive

to quantization. For example, we observe a significant accuracy drop on MobileNet-V1 for $B_{w,\text{eff}} \leq 3$ when using Deep Compression [86] and HAQ [82]. In contrast to HAQ and Deep Compression, IMPQ is effective even at a lower precision, e.g., the accuracy of Deep Compression and HAQ at $B_{w,\text{eff}} = 3$ is comparable to the accuracy of IMPQ at $B_{w,\text{eff}} = 2$. Thus IMPQ leads to a 33% better compression on MobileNet-V1 for weight-only quantization.

Table 2.1: Classification accuracy on CIFAR-10 using ResNet-20 with weight-only quantization.

Method	$B_{w,\text{eff}}$	Baseline	Accuracy	change
BWN [87]	1	92.10	90.2	1.90
TWN [66]	Ternary	91.77	90.78	0.89
TTQ [67]	Ternary	91.77	91.13	0.64
ELQ [77]	Ternary	91.25	91.45	-0.20
ELQ [77]	1	91.25	91.15	0.10
DoReFa [68]	3	92.10	91.81	0.29
DoReFa [68]	2	92.10	91.41	0.69
LQ-Net [78]	3	92.00	92.00	0
LQ-Net [78]	2	92.00	91.80	0.20
IMPQ	1.74	92.10	92.00	0.10

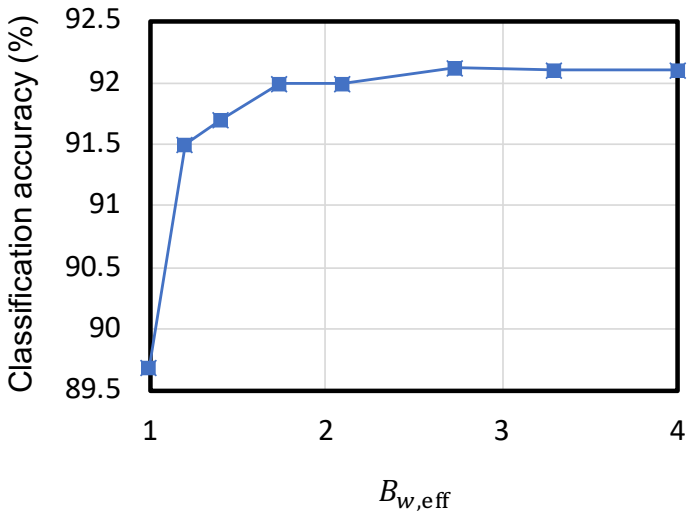


Figure 2.2: Classification accuracy on CIFAR-10 using ResNet-20 as a function of the effective precision recorded at the end of each iteration.

Table 2.2: Classification accuracy on CIFAR-10 using VGG-Small with weight-only quantization.

Method	$B_{w,\text{eff}}$	Baseline	Accuracy	change
BWN [87]	1	93.18	91.77	1.45
TWN [66]	Ternary	93.18	92.56	0.62
LQ-Net [78]	2	93.8	93.8	0
IMPQ	1.55	93.1	92.97	0.13

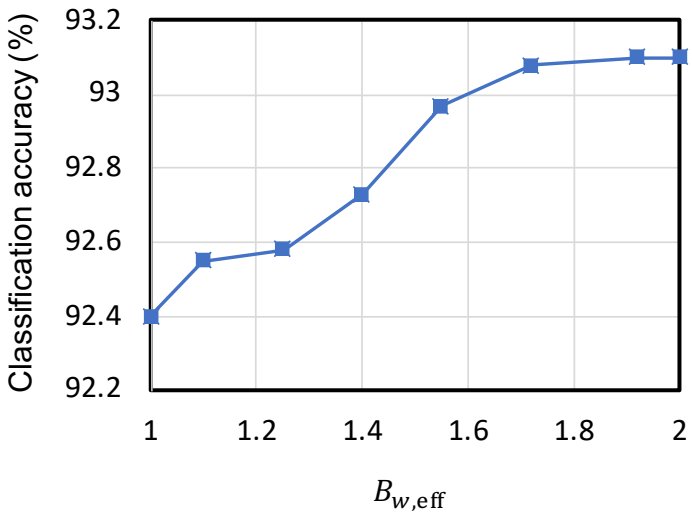


Figure 2.3: Classification accuracy on CIFAR-10 using VGG-Small as a function of the effective precision recorded at the end of each iteration.

Impact of Granularity: We hypothesize that a granular assignment of precision would lead to better network compression on the whole. We study this impact of granularity using the compression ratio (CR) as a metric, defined as:

$$\text{CR} = \frac{B \sum_k \mathcal{W}_k}{\sum_k B_{w,k} \mathcal{W}_k} \tag{2.10}$$

where B is the precision of a baseline network we are comparing with. The compression ratio quantifies the reduction in model size for storage with respect to a baseline network. We assume that the baseline network to use 16-b weights. We applied IMPQ with: 1) layer-wise, 2) kernel-wise, and 3) network-wise precision allocation. We find that kernel-wise precision allocation gave the best compression ratio followed by layer-wise precision allocation, thereby supporting our hypothesis (see Fig. 2.4).

Table 2.3: Classification accuracy on SVHN using VGG-Small with weight-only quantization.

Method	$B_{w,\text{eff}}$	Baseline	Accuracy	change
BWN [87]	1	97.54	96.22	-1.32
LQ-Net [78]	2	97.54	97.62	-0.08
IMPQ	1.7	97.54	97.58	-0.04

Table 2.4: Classification accuracy on ImageNet-1k using MobileNet-V1 with weight-only quantization.

Method	$B_{w,\text{eff}}$	Baseline	Accuracy	change
DeepC [82, 86]	2	70.90	37.62	-33.28
DeepC [82, 86]	3	70.90	65.94	-4.96
DeepC [82, 86]	4	70.90	71.14	0.24
HAQ [82]	2	70.90	57.14	-13.76
HAQ [82]	3	70.90	67.66	-3.24
HAQ [82]	4	70.90	71.74	0.84
IMPQ	2	71.20	66.51	-4.71
IMPQ	3	71.20	68.3	-2.92
IMPQ	4	71.20	70.2	-2.02

Trends across layers: Figure 2.5 shows the layer-wise effective weight precision of MobileNet-V1 quantized using IMPQ. We observe the following trends: a) fully-connected layers that constitute 25% of the parameters are aggressively quantized, b) layers closer to the input image are the most sensitive and hence quantized less, and c) point-wise layers that have more parameters have fewer bits than the depth-wise layers. In general, we find that the layers with more parameters and farther from the input are less sensitive, and hence quantized more heavily.

2.4.2 Weight and Activation Quantization

For the simultaneous quantization of both weights and activations, we first apply IMPQ with weight-only quantization, and then we extend to activation quantization. Note that applying the activation quantization first is also possible. Since weight quantization trades off with the activation quantization, choosing aggressively quantized baseline will limit the extent of acti-

Table 2.5: Classification accuracy on ImageNet-1k using MobileNet-V1 with both weights and activations quantized.

Method	$B_{w,\text{eff}}$	$B_{a,\text{eff}}$	Accuracy
PACT [69, 82]	6	6	71.22
PACT [69, 82]	5	5	67.00
PACT [69, 82]	4	4	62.44
HAQ [82]	6	6	70.90
HAQ [82]	5	5	70.58
HAQ [82]	4	4	67.40
UNIQ [88]	8	8	68.25
UNIQ [88]	5	8	67.5
UNIQ [88]	4	8	66.0
RQ [89]	6	6	67.50
RQ [89]	4	4	61.50
IMPQ	6	6	71.24
IMPQ	5	5	69.97
IMPQ	4	4	69.02

vation precision reduction. Activations are quantized layer-wise so that the dot-product computations of the networks can be mapped to a fixed-point hardware. Table 2.5 summarizes the accuracy and effective precision of both weights and activations compared to other recent works. The techniques that use identical precision across the network, such as UNIQ, RQ, and PACT, result in significant accuracy drop with precision reduction. For example, PACT observes $> 7\%$ accuracy drop going from a 6-b quantization to a 4-b quantization. This accuracy drop with layer-wise quantization techniques such as HAQ is 3.5%, while IMPQ's kernel-wise quantization results in an accuracy drop of 2.22%. We find that IMPQ demonstrates the best accuracy with $B_{w,\text{eff}} = B_{a,\text{eff}} = 4$.

2.5 Conclusions

This chapter presents the IMPQ methodology to obtain a mixed-precision quantized network with precision allocated at the kernel-wise granularity. IMPQ was validated with experiments on ResNet-20, VGG-Small for classifying the CIFAR-10 dataset, VGG-Small for classifying the SVHN dataset, and MobileNet-V1 for classifying the ImageNet-1k dataset. IMPQ was found

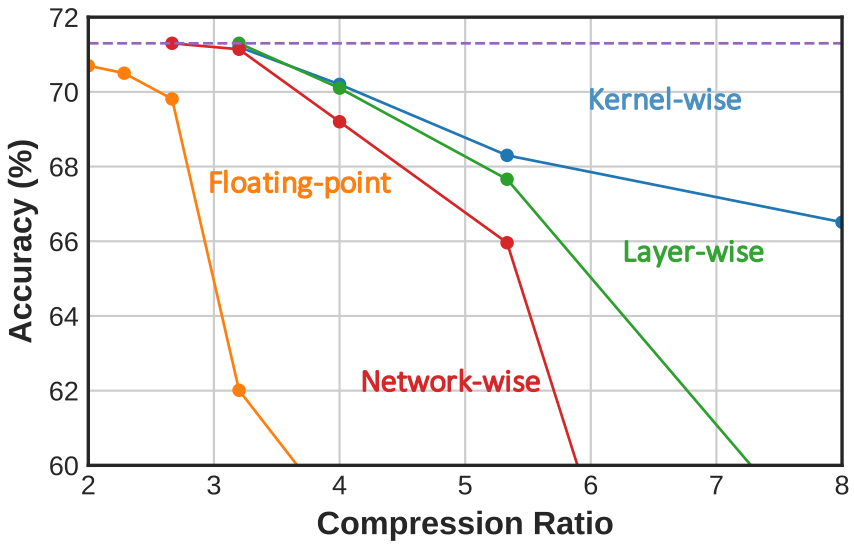


Figure 2.4: Classification accuracy on ImageNet-1k using MobileNet-V1 vs. compression ratio (CR) and the granularity of precision assignment. Accuracy of floating-point network is with no retraining. Compression ratio is calculated with respect to a fixed point network with 16-b weights.

to be most effective on compact networks and at lower precision compared to other state-of-the-art schemes.

The experiments show that in most compact networks, majority of computations require $<$ 5-b precision for weights and inputs. These trends bode well for the use of DIMA since analog computations are more energy efficient that digital for lower precisions ($<$ 6-b) [59]. The compositional framework presented in Chapter 3 can be used to design such in-memory architectures and the analysis presented in Chapter 4 can be used to meet the precision requirements of the DNNs.

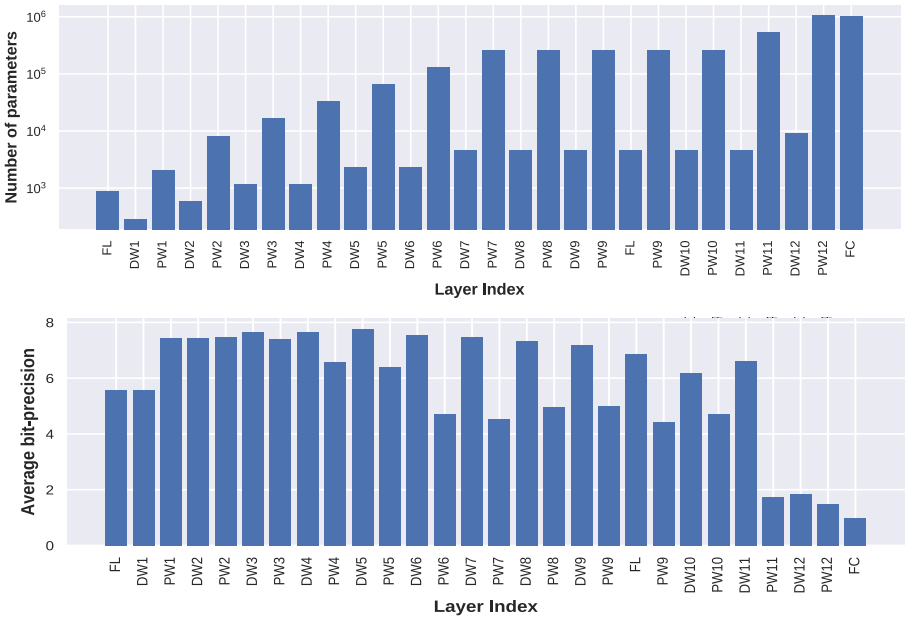


Figure 2.5: Effective weight precision across different layers of MobileNet-V1 used for classifying the ImageNet-1k dataset after the application of IMPQ for weight-only quantization.

CHAPTER 3

A COMPOSITIONAL FRAMEWORK FOR IN-MEMORY ARCHITECTURES

Understanding the relationship between energy consumption, SNR, and latency of DIMA is essential to design architectures that meet system-level accuracy and energy requirements. However, the cross-layered nature of DIMA makes it challenging to predict this relationship as the design choices span the entire compute stack – memory devices, bitcell circuit topologies, architectural options, and others.

To address this challenge, this chapter proposes a *compositional framework* for designing in-memory architectures that incorporates: (1) three *signal-to-noise ratio (SNR) metrics* (analog SNR ($\mathrm{SNR_a}$), digitization SNR ($\mathrm{SNR_d}$), and total SNR ($\mathrm{SNR_T}$)); (2) three *in-memory compute models* (charge accumulation (QA), current accumulation (QA) and charge sharing (QS)); and (3) four *mixed-signal arithmetic decomposition* methods (MM, BM, MB, BB) denoting the binarization of weights and/or activations and partitioning of a *multi-bit dot-product* functionality into analog and digital computations. This compositional framework enables designers to: (1) explain the functionality, trade-offs and limits on energy-efficiency and accuracy of a large class of recently published in-memory IC prototypes listed in Table 3.1, and (2) provide guidelines for designing new in-memory architectures.

3.1 SNR Metrics for In-memory Architectures

In this section, we propose SNR metrics for quantifying the computational accuracy of in-memory architectures implementing the following dot-product:

$$y_o = \mathbf{w}^\mathsf{T}\mathbf{x} = \sum_{j=1}^{N} w_j x_j \tag{3.1}$$

Table 3.1: Classification of in-memory architectures based on the compute models.

		In-memory Compute Model			Precision in Analog	
		QA	IA	QS	B_x	B_w
CMOS	Kang *et al.* [28]	✓		✓	8	8
	Biswas *et al.* [29]			✓	8	1
	Zhang *et al.* [30]	✓			5	1
	Valavi *et al.* [31]			✓	1	1
	Khwa *et al.* [32]		✓		1	1
	Jiang *et al.* [33]		✓		1	1
	Si *et al.* [34]	✓		✓	2	5
	Jia *et al.* [35]			✓	1	1
	Okumura *et al.* [36]		✓		1	T
	Kim *et al.* [37]		✓		1	1
	Guo *et al.* [38]	✓			1	1
	Yue *et al.* [39]	✓			2	5
	Su *et al.* [40]	✓			2	1
	Dong *et al.* [41]	✓		✓	4	4
	Si *et al.* [42]	✓			2	2
Beyond CMOS	Chen *et al.* [43]		✓		1	T
	Fick *et al.* [44]		✓		A	A
	Xue *et al.* [45]		✓		1	1
	Yan *et al.* [46]		✓		1	1
	Zha *et al.* [47]	✓			1	1
	Xue *et al.* [48]		✓		2	4

T: Ternary; A: Analog/continuous-valued variable

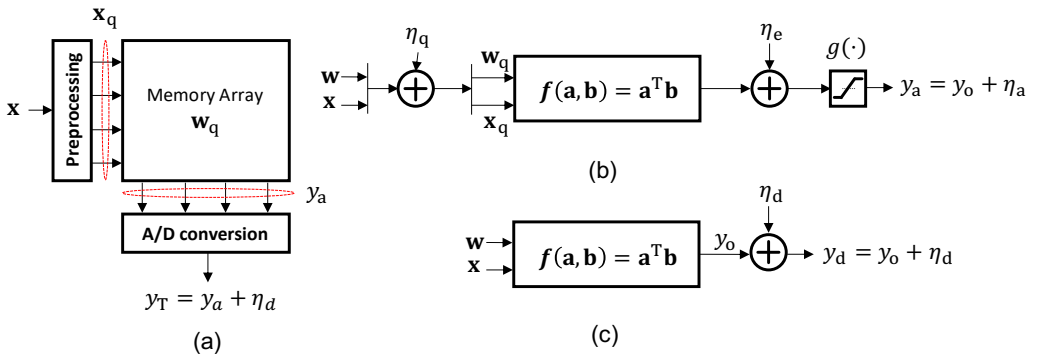


Figure 3.1: Noise sources contributing to dot-product SNR of in-memory architectures: (a) total SNR (SNR_T), (b) analog SNR (SNR_a), and (c) digitization SNR (SNR_d).

where y_o is the DP of two N-dimensional real-valued vectors $\mathbf{w} = [w_1, \ldots, w_N]^\mathsf{T}$ (weight vector) and $\mathbf{x} = [x_1, \ldots, x_N]^\mathsf{T}$ (input/activation vector) of precision B_w and B_x, respectively. The dot-product is a key computational kernel in DNNs. A digital architecture realizes (3.1) via N $B_x \times B_w$-b multiply-accumulate (MAC) operations with quantization noise as the primary source of its non-ideal behavior.

In-memory architectures (Fig. 3.1) exhibit four dominant noise sources when compared to a floating-point baseline:

- data (weight and activation) quantization noise η_q (Fig. 3.1(b)),
- clipping due to the limited dynamic range of the BL modeled as the non-linearity $g()$ (Fig. 3.1(b)),
- circuit non-idealities such as spatial variations in transistor threshold voltage V_t and others η_e (Fig. 3.1(b)), and
- quantization noise η_d due to finite analog-to-digital converter (ADC) resolution (Fig. 3.1(c)). It depends upon the ADC input swing and the ADC precision B_y.

The presence of these noise sources is described by the following expressions:

$$y_{\mathrm{a}} = y_{\mathrm{o}} + \eta_{\mathrm{a}} = y_{\mathrm{o}} + \eta_{\mathrm{q}\to y} + \eta_{\mathrm{e}} + \eta_{\mathrm{c}} \tag{3.2}$$

$$\eta_{\mathrm{c}} = g(y_a) - (y_o + \eta_{\mathrm{q}\to y} + \eta_{\mathrm{e}}) \approx g(y_{\mathrm{o}}) - y_{\mathrm{o}} \tag{3.3}$$

$$y_{\mathrm{d}} = y_{\mathrm{o}} + \eta_{\mathrm{d}} \tag{3.4}$$

$$y_{\mathrm{T}} = y_{\mathrm{a}} + \eta_{\mathrm{d}} \tag{3.5}$$

where y_{a} is the pre-ADC output, y_{o} is the ideal dot-product value as defined in (3.1), η_{a} is the total pre-ADC noise that includes the data quantization noise η_{q} reflected at the ADC input as $\eta_{\mathrm{q}\to y}$, η_{c} is the clipping noise term, y_{d} is the dot-product value incorporating only the ADC quantization noise η_{d}, and y_{T} (Fig.. 3.1(a)) is the final dot-product value incorporating all the four noise sources η_{q}, η_{c}, η_{e}, and η_{d}.

Equations (3.2)-(3.5) lead to the definitions of three SNR metrics (see Fig. 3.1): (1) the *analog SNR* ($\mathrm{SNR_a}$), (2) the *digitization SNR* ($\mathrm{SNR_d}$), and (3) *total SNR* ($\mathrm{SNR_T}$) defined as:

$$\mathrm{SNR_a} = \frac{\sigma^2_{y_{\mathrm{o}}}}{\sigma^2_{\eta_{\mathrm{a}}}} = \frac{\sigma^2_{y_{\mathrm{o}}}}{\sigma^2_{\eta_{\mathrm{q}\to y}} + \sigma^2_{\eta_{\mathrm{c}}} + \sigma^2_{\eta_{\mathrm{e}}}} \tag{3.6}$$

$$\mathrm{SNR_d} = \frac{\sigma^2_{y_{\mathrm{o}}}}{\sigma^2_{\eta_{\mathrm{d}}}} \tag{3.7}$$

$$\mathrm{SNR_T} = \frac{\sigma^2_{y_{\mathrm{o}}}}{\sigma^2_{\eta_{\mathrm{a}}} + \sigma^2_{\eta_{\mathrm{d}}}} = \left[\frac{1}{\mathrm{SNR_a}} + \frac{1}{\mathrm{SNR_d}}\right]^{-1} \tag{3.8}$$

where $\sigma^2_{\eta_{\mathrm{a}}}$, $\sigma^2_{\eta_{\mathrm{q}\to y}}$, $\sigma^2_{\eta_{\mathrm{c}}}$, $\sigma^2_{\eta_{\mathrm{e}}}$, and $\sigma^2_{\eta_{\mathrm{d}}}$ are the variances of η_{a}, $\eta_{\mathrm{q}\to y}$, η_{c}, η_{e}, and η_{d}, respectively. Note that $\sigma^2_{\eta_{\mathrm{c}}}$ and $\sigma^2_{\eta_{\mathrm{e}}}$ depend upon the specifics of the in-memory architecture being considered. In contrast, $\sigma^2_{y_{\mathrm{o}}}$ and $\sigma^2_{\eta_{\mathrm{q}\to y}}$ depend only upon algorithmic parameters and the precision, and are given by:

$$\sigma^2_{y_{\mathrm{o}}} = N\sigma^2_w \mathbb{E}[x^2]x^2 \tag{3.9}$$

$$\sigma^2_{\eta_{\mathrm{q}\to y}} = \frac{N}{12}\left(\Delta^2_w \mathbb{E}[x^2] + \Delta^2_x \sigma^2_w\right) \tag{3.10}$$

where σ^2_w is the variance of the weights, $\Delta_w = 2^{-B_w+1}$, and $\Delta_x = 2^{-B_x}$ are the weight and input quantization step-sizes, respectively.

Equation (3.8) indicates that the smaller of $\mathrm{SNR_a}$ and $\mathrm{SNR_d}$ determines $\mathrm{SNR_T}$. Since $\mathrm{SNR_d}$ increases with ADC precision B_y, it is $\mathrm{SNR_a}$ that funda-

mentally limits $\mathrm{SNR_T}$. Therefore, when exploring the limits of precision of in-memory architectures, we first determine an upper bound on $\mathrm{SNR_a}$ and then determine a suitable ADC precision B_y such that $\mathrm{SNR_T}$ is close to $\mathrm{SNR_a}$, i.e., B_y is chosen to minimize $\mathrm{SNR_a} - \mathrm{SNR_T} \leq \Delta\mathrm{SNR}$. In this chapter, without any loss of generality, we choose $\Delta\mathrm{SNR} = 0.5\,\mathrm{dB}$ which would result in an increase of $1.12\times$ in the DNN misclassification rate [70, 73]. This implies that $\mathrm{SNR_d} \geq \mathrm{SNR_a} + 9$ (dB) from which an appropriate ADC precision B_y can be determined.

3.2 In-memory Compute Models

An in-memory compute model is a mapping of algorithmic variables y_o, x_j and w_j in (3.1) to circuit variables such as time, charge, current, or voltage in order to (usually partially) realize an analog BL computation of the multi-bit dot-product in (3.1). In practice, most in-memory architectures employ (usually) one or (sometimes) two of these compute models along with additional digital computation to realize the $B_w \times B_x$-b N-dimensional dot-product in (3.1) as summarized in Table 3.1.

In this section, we propose three in-memory compute models that underlie recently published in-memory ICs: (a) *charge accumulation* (QA) [27, 28, 30, 49]; (b) *current accumulation* (IA) [32–34, 37]; and (c) *charge sharing* (QS) [28, 29, 31, 49]. We conjecture that these compute models are in some sense universal in that they represent an approximation to a 'complete set' of practical, i.e., realizable, variable mappings from the algorithmic to the circuit domain in the context of in-memory architectures.

This section provides analytical expressions for circuit domain equivalents of η_e and η_c in (3.2)-(3.3) for each compute model. These will be combined with algorithm and precision-dependent noise sources η_q and η_d in Section 3.3 to obtain $\mathrm{SNR_T}$. Table 3.2 tabulates the parameters of all three in-memory compute models in a representative 65 nm CMOS process.

3.2.1 The Charge Accumulation (QA) Model

The QA model (see Fig. 3.2(a)) realizes the dot-product in (3.1) by integrating the cell current I_j over a WL pulse duration T_j ($j = 1, \ldots, N$) on a BL

(or cell) capacitor C resulting an output voltage given by:

$$V_{\rm o} = \frac{1}{C}\sum_{j=1}^{N} I_j T_j \tag{3.11}$$

where $V_{\rm o}$ is the dot-product output assuming infinite voltage head-room, i.e., no clipping. In other words, the QA model is defined by the mapping of variables in (3.1) as: $y_{\rm o} \rightarrow V_{\rm o}$, $w_j \rightarrow I_j$, and $x_j \rightarrow T_j$. As seen in Table 3.2, typical parameters values of the QA model are: C (a few hundred fFs), I_j (tens of μAs), and T_j (hundreds of ps).

Noise Models: The noise contributions in QA arise from the following sources: (1) variations in the pulse-widths T_j of current switch pulses ϕ_j (Fig. 3.2(a)); (2) their finite rise and fall times (see Fig. 3.3(b)); (3) spatial variations in the currents I_j; (4) thermal noise in the discharge RC-network; and (5) clipping due to limited voltage head-room. Thus, the dot product output $V_{\rm a}$ corresponding to $y_{\rm a}$ in (3.2) is given by:

$$\begin{aligned} V_{\rm a} &= V_{\rm o} + v_{\rm e} + v_{\rm c}, \\ v_{\rm e} &= v_\theta + \frac{1}{C}\sum_{j}^{N}(I_j + i_j)(T_j + t_j - t_{\rm rf}) - V_{\rm o}, \\ v_{\rm c} &= \min\left(V_{\rm o},\ V_{\rm o,max}\right) - V_{\rm o}, \end{aligned} \tag{3.12}$$

where $V_{\rm o,max}$ is the maximum allowable output voltage, and $v_{\rm e}$ and $v_{\rm c}$ are the voltage domain noise due to circuit non-idealities and clipping, respectively, corresponding to the noise terms $\eta_{\rm e,c}$ in (3.2), $i_j \sim \mathcal{N}(0, \sigma_{I_j}^2)$ and $t_j \sim \mathcal{N}(0, \sigma_{T_j}^2)$ are the noise due to (spatial) current mismatch and (temporal) pulse-width mismatch, respectively, both of which are modeled as zero mean Gaussian random variables, $t_{\rm rf}$ models the impact of finite rise and fall times of the current switching pulses, and $v_\theta \sim \mathcal{N}(0, \sigma_\theta)$ is the thermal noise. Note: $V_{\rm o,max}$ can be as high as 0.9 V with a $V_{\rm dd} = 1$ V.

Analytical expressions to estimate the noise standard deviations σ_{Ij}, σ_{T_j},

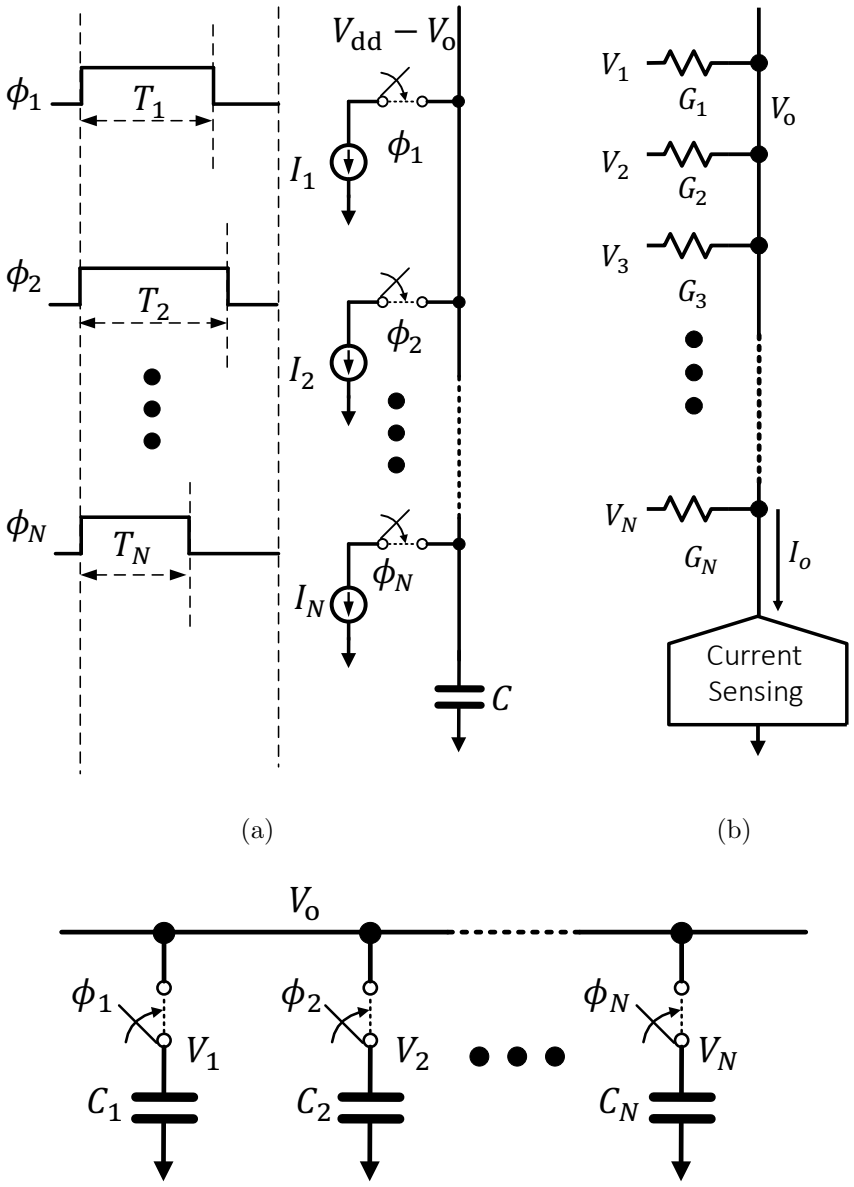
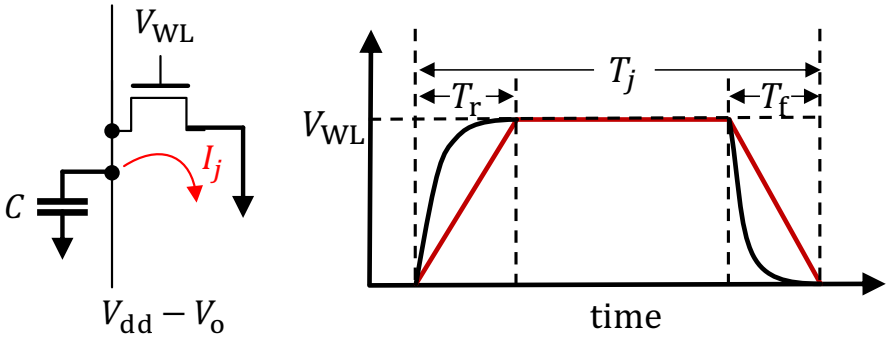


Figure 3.2: In-memory compute models: (a) charge accumulation (QA), (b) current accumulation (IA), and (c) charge sharing (QS).



(a) (b)

Figure 3.3: Modeling the discharge process in the QA compute model: (a) word-line voltage pulse V_{WL}, and (b) cell current I_j.

t_{rf}, and σ_θ (derived at the end of the chapter) are provided below:

$$\sigma_{I_j} = I_j\left(\frac{\alpha\sigma_{V_\mathrm{t}}}{V_{\mathrm{WL}} - V_\mathrm{t}}\right) \tag{3.13}$$

$$\sigma_{T_j} = \sqrt{h_j}\sigma_{T0} \tag{3.14}$$

$$t_{\mathrm{rf}} = T_r - \left(\frac{V_{\mathrm{WL}} - V_\mathrm{t}}{V_{\mathrm{WL}}}\right)\frac{T_\mathrm{r} + T_\mathrm{f}}{\alpha + 1} \tag{3.15}$$

$$\sigma_\theta = \sqrt{\frac{kT}{C}} \tag{3.16}$$

where $T_j = h_j T_0$, T_0 is the delay of a unit element cascaded h_j times to obtain a WL pulse-width of T_j, σ_{T0} is the standard deviation of T_0, T_r and T_f are WL pulse rise and fall times (see Fig. 3.3(b)), α is a fitting parameter used for an α-law transistor current equation, σ_{V_t} is standard deviation of V_t variations, k is the Boltzmann constant, and T is the absolute temperature. Note that the cell current I_j depends upon the transistor sizes and the WL voltage V_{WL}.

Note that typically the WL voltage V_{WL} is identical for all rows in the memory array with a few exceptions such as [30] which modulate V_{WL} to tune the cell current I_j. The effects of rise and fall time, and delay variations, can be mitigated by carefully designing the WL pulse generators. The noise in QA is dominated by threshold voltage spatial variations. Indeed, using the typical values from Table 3.2, we find that σ_{Ij}/Ij ranges from 8% to 25%, while σ_{Tj}/Tj ranges from 0.5% to 3%.

Energy and Delay Models: The average energy consumption in the QA

model is given by:

$$E_{\mathrm{QA}} = \mathbb{E}\left[V_{\mathrm{a}}\right] V_{\mathrm{dd}} C + E_{\mathrm{su}} \tag{3.17}$$

where E_{su} represents the energy consumed in toggling the switches ϕ_js. Equation (3.17) shows that the energy consumption in the QA model increases with $C \propto$ array size, the supply voltage V_{dd}, and the mean value of the dot-product $\mathbb{E}\left[V_{\mathrm{a}}\right]$.

The delay of the QA model is given by:

$$T_{\mathrm{QA}} = T_{\mathrm{max}} + T_{\mathrm{su}} \tag{3.18}$$

where T_{su} is the time required to precharge the capacitors and setup currents, and $T_{\mathrm{max}} = \max\{T_j\}$ is the longest allowable pulse-width.

3.2.2 The Current Accumulation (IA) Model

The IA model (Fig. 3.2(b)), commonly employed resistive crossbar arrays [43, 45] and more recently in CMOS-based in-memory architectures, e.g., [32–34, 37], realizes the dot-product in (3.1) by driving a resistive path of conductance G_j with voltage inputs V_j ($j = 1, \ldots, N$) to obtain the output current I_{o} as:

$$I_{\mathrm{o}} = \sum_{j=1}^{N} I_j = \sum_{j=1}^{N} (V_j - V_{\mathrm{o}}) G_j \tag{3.19}$$

where V_{o} is the output voltage. The IA model is defined by the mapping of the variables in (3.1) as follows: $y_{\mathrm{o}} \rightarrow I_{\mathrm{o}}$, $w_j \rightarrow G_j$, and $x_j \rightarrow (V_j - V_{\mathrm{o}})$. Typical parameters values of the IA model are: G_j (from a few $\mathrm{k}\Omega^{-1}$s (MRAM) [90] to hundreds of $\mathrm{k}\Omega^{-1}$ (RRAM) [91]), and V_j (couple of hundred mV).

Unlike the QA model, the IA model does not suffer from clipping due dynamic range limitations of the BL. However, clipping can still occur if I_{o} exceeds the input dynamic range of the current sensing circuits in the periphery of the array. Some in-memory architectures [32, 33, 37] employ voltage sensing where the voltage V_{o} is permitted to change and used for sensing opposed to sensing I_{o}. In these architectures, a severe non-linearity is observed due to the dependence between the G_j and V_{o}, thereby requiring

non-linear A/D conversion at the output [33].

Noise Models: Noise contributions in IA arise from: (1) spatial conductance variations; (2) write noise; and (3) thermal noise currents in the resistors. We will assume that the IA compute model does not clip. Hence, the dot product output I_{a} corresponding to y_{a} in (3.2) is given by:

$$I_{\mathrm{a}} = I_{\mathrm{o}} + i_{\mathrm{e}} = \sum_{j} \big(V_j - V_{\mathrm{o}}\big)\big(G_j + g_j\big) + i_{\theta_j} \tag{3.20}$$

where i_{e} is the current domain noise term due to circuit non-idealities corresponding to the term η_{e} in (3.2), $g_j \sim \mathcal{N}(0, \sigma^2_{G_j})$ accounts for the write noise, spatial variations and mismatch, and $i_{\theta j} \sim \mathcal{N}(0, \sigma^2_{\theta j})$ is the thermal and shot noise current components whose standard deviation is given by:

$$\sigma_{\theta_j} = \sqrt{(2q(V_j - V_{\mathrm{o}}) + 4kT)(T_{\mathrm{IA}})^{-1} G_j} \tag{3.21}$$

where T_{IA} is the current sensing period, and q is the electron charge. Typically spatial variations and write noise components dominate the overall SNR, e.g., MRAM spatial variations can be has high as 7% [92] and the write noise in ReRAM can be as high as 4% [91].

Energy and Delay Models: The average energy consumption in the IA model is given by:

$$E_{\mathrm{IA}} = \mathbb{E}\left[I_{\mathrm{o}}\right] V_{\mathrm{dd}} T_{\mathrm{IA}} + E_{\mathrm{su}} + E_{\mathrm{sense}} \tag{3.22}$$

where E_{su} represents the energy costs associated with establishing the voltages and charging internal node capacitances, and E_{sense} is the energy needed by the sensing (readout) circuits. Equation (3.22) shows that the energy consumption in the IA model increases with the sensing period T_{IA}, the supply voltage V_{dd}, and the mean value of the dot-product $\mathbb{E}\left[I_{\mathrm{o}}\right]$.

The delay of the IA model depends on the sensing scheme and the resolution of the outputs required. Typically the output currents are integrated an a sampling capacitance before digital conversion [93]. The minimum delay

of the IA model $T_{\rm IA}$, is given by:

$$T_{\rm IA} = T_{\rm sense} + T_{\rm su} \tag{3.23}$$
$$T_{\rm sense} = \frac{V_{\rm dd} C_{\rm sample}}{I_{\rm o,max}}$$

where $T_{\rm sense}$ is the sampling time required for sensing, $T_{\rm su}$ is the time required to establish the output voltage $V_{\rm o}$ and for the currents to stabilize, $C_{\rm sample}$ is the sampling capacitance and $I_{\rm o,max}$ is the maximum output current. For an 8-b resolution, sampling capacitances in the order of 10 fF are required.

Comparing the first terms in (3.17) and (3.22) we find that IA is efficient with larger arrays and where the BL capacitance are larger. However, these benefits will diminish with increasing sensing resolution requirements.

3.2.3 The Charge Sharing (QS) Model

The CS model (Fig. 3.2(c)) is commonly employed to perform the additions in (3.1). The multiplications in (3.1) are separately computed via charging/discharging capacitor C_j in proportion to the product $w_j x_j$ ($j = 1, \ldots, N$) as in [29, 31], or by employing explicit multiplier circuits such as in [28, 49]. The N capacitors share charge via a sequence of switching events (see Fig. 3.2(c)) to generate the final voltage $V_{\rm o}$ given by:

$$V_{\rm o} = \frac{1}{\sum_j C_j} \sum_j C_j V_j \tag{3.24}$$

The CS model is defined by the mapping of the variables in (3.1) as follows: $y_{\rm o} \rightarrow V_{\rm o}$, and $w_j x_j \rightarrow V_j$. The capacitors C_j are typically identical metal-on-metal (MOM) capacitors with values ranging from 1 fF to 10 fF [28, 31].

Noise Models: Assuming metal-on-metal (MOM) capacitor to realize C_js, the noise contributions in QS arise from: (1) capacitor mismatch [94]; (2) charge injection due to switching [95]; and (3) thermal noise. Unlike QA, and similar to IA, the QS model does not suffer from clipping noise and distortion. Hence, the dot-product output $V_{\rm a}$ corresponding to $y_{\rm a}$ in (3.2) is

given by:

$$\begin{aligned} V_{\mathrm{a}} &= V_{\mathrm{o}} + v_{\mathrm{e}} \\ &= \frac{1}{\sum_j (C_j + c_j)} \sum_j (C_j + c_j)(V_j + v_{\theta_j} + v_j) \end{aligned} \tag{3.25}$$

where v_{e} is the voltage domain noise term due to circuit non-idealities corresponding to the η_{e} in (3.2), v_j is the noise is due to charge injection, $c_j \sim \mathcal{N}(0, \sigma^2_{C_j})$ is the capacitor mismatch, and $v_{\theta_j} \sim \mathcal{N}(0, \sigma_{\theta,j})$ is the thermal noise. Furthermore, the expressions for the noise parameters in (3.25) can be derived as:

$$\sigma_{C_j} = \kappa \sqrt{C_j} \tag{3.26}$$

$$\delta V_j = p \frac{WLC_{\mathrm{ox}}(V_{\mathrm{dd}} - V_{\mathrm{t}} - V_j)}{C_j} \tag{3.27}$$

$$\sigma_\theta = \sqrt{\frac{kT}{C_j}} \tag{3.28}$$

where κ is a technology- and layout-dependent Pelgrom coefficient [94], $0 \leq p \leq 1$ is constant that depends on the layout of the switch transistor, C_{ox} is the gate oxide capacitance per unit area, and W and L are the width and length of the switch transistor. The effect of noise in the QS compute model can be minimized by increasing the capacitors sizes at the expense of energy consumption as seen from (3.29) below.

Energy and Delay Models: The average energy consumption in the QS model is given by:

$$E_{\mathrm{QS}} = \sum_j \mathbb{E}\left[(V_{\mathrm{dd}} - V_j)\right] V_{\mathrm{dd}} C_j + E_{\mathrm{su}} \tag{3.29}$$

where E_{su} includes energy cost for the switches ϕ_js.

The delay of the QS model is given by:

$$T_{\mathrm{QS}} = T_{\mathrm{share}} + T_{\mathrm{su}} \tag{3.30}$$

where T_{share} is the time required for charge sharing to complete, and T_{su} is the time required to precharge the capacitors to the desired voltages V_j.

Table 3.2: Estimated parameters for a representative 65 nm CMOS process

	Parameter	Value	Parameter	Value
QA	k'	$220\,\mu\text{A/V}^2$	α	1.8
	σ_{T0}	2.3 ps	σ_{V_t}	23.8 mV
	$\Delta V_{\text{BL,max}}$	0.8 V-0.9 V	V_{WL}	$0.4\,\text{V} - 0.8\,\text{V}$
	V_t	0.4 V	T_0	100 ps
IA	q	1.6e-19 C	σ_{G_j}	1%-10%
QS	WLC_{ox}	0.31 fF	κ	$0.08\,\text{fF}^{0.5}$
	p	0.5		
Common Parameters				
	T	270 K	k	$1.38\text{e-}23\,\text{JK}^{-1}$
	V_{dd}	1 V		

3.2.4 Summary of In-memory Compute Models

Table 3.2 tabulates the parameters of all three in-memory compute models in a representative 65 nm CMOS process. The in-memory compute models in this section describe the realization of an analog BL computation of a multi-bit dot-product (3.1) by mapping the inputs x_j, weights w_j and the output y_o to physical variables such as charge, current, voltage, and capacitance. Since the multi-bit dot-product computation in (3.1) is high-dimensional, it is clear that the dynamic range and therefore SNR limits imposed by the analog nature of BL computations will be reached at some point. It is for this reason that most, if not all, in-memory architectures resort to some form of arithmetic-level decomposition which partially binarizes the multi-bit dot-product in (3.1) in order to limit the BL dynamic range and then employs one or more of the in-memory compute models, the exception being CM [28, 49], which limits the BL dynamic range by sequentially processing dot-products. In either case, the dynamic range and hence SNR limits the number and precision of BL computations per read cycle and hence the overall energy-efficiency of in-memory architectures.

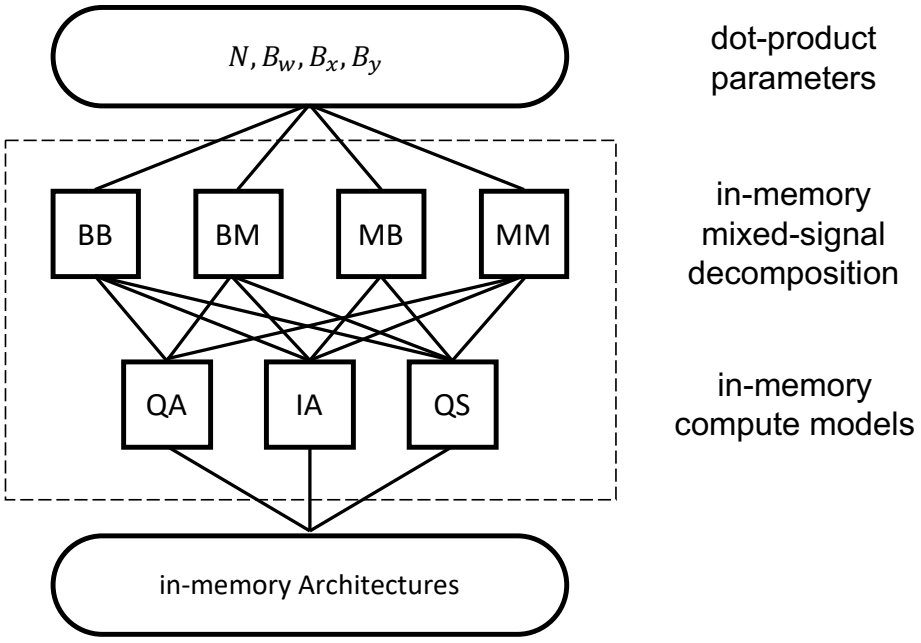


Figure 3.4: The proposed compositional framework for in-memory architectures.

3.3 A Compositional Framework for In-memory Architectures

In this section, we present a compositional framework (see Fig. 3.4) that employs mixed-signal arithmetic decomposition methods to binarize the multi-bit dot-product in (3.1) and then employs one or more of the in-memory compute models from Section 3.2 to construct a variety of in-memory architectures. In doing so, the compositional framework offers a systematic way to compare and analyze current and future in-memory architectures in terms of their fundamental energy-efficiency and accuracy trade-offs.

Though the framework in Fig. 3.4 indicates that there are twelve possible in-memory architectures, not all will be attractive due to intrinsic circuit limitations. For example, though all four of QA-(BB,BM,MM,MB) architectures are all possible the QA-BM, QA-MM, QA-MB architectures may not be attractive as they severely restrict the output precision B_y and/or the dot-product length N. Similarly, architectures IA-BM, IA-BM, and IA-MB are all realizable. However, IA-BM, IA-MB and IA-MM require WL access pulses that are amplitude-modulated, which can be challenging to realize due to the non-linear relationship between the WL voltages and the BC currents.

Finally, the architectures QS-BB, QS-BM are possible in principle, but QS-MM and QS-MB are not possible unless an explicit multiplier is employed or another compute model is used. Overall, 8 of 12 architectures are feasible in practice with some being more challenging than others.

In this section, we first present the four mixed-signal decomposition methods (MM, BM, MB, and BB) followed by the construction of six architectures that are deemed to be practical, viz. QA-BB, QA-BM, QA-MB, QS-BB, QS-BM, and CM, which is a composite architecture employing both the QA and QS compute models. In the following, we refer to computations in which both the weight and activations are binary-valued as *binarized*, and those with binary-valued weights and multi-bit inputs as *binary-weighted*. Furthermore, we employ the term *powers-of-two summing* to refer to the summing of multiple variables which are weighed with powers-of-two coefficients as in (3.31)–(3.32).

3.3.1 Mixed-signal Arithmetic Decomposition

Any N-dimensional vector $\mathbf{w}$ of B_w-b unsigned scalars w_j $(j = 1, \ldots, N)$ can be represented as the following powers-of-two sum:

$$\mathbf{w} = \sum_{i=1}^{B_w} 2^{-i} \hat{\mathbf{w}}_i \tag{3.31}$$

where $\hat{\mathbf{w}}_i \in \{0, 1\}$ is a N-dimensional binary vector comprising the i-th bit $\hat{w}_{i,j}$ of w_j, i.e.,

$$w_j = \sum_{i=1}^{B_w} 2^{-i} \hat{w}_{i,j} \tag{3.32}$$

Different number representations, e.g., signed, will result in a different mapping from a bit-vector representation to the numerical value of the scalar w_j.

Mixed-signal decomposition of the multi-bit dot-product (3.1) is defined

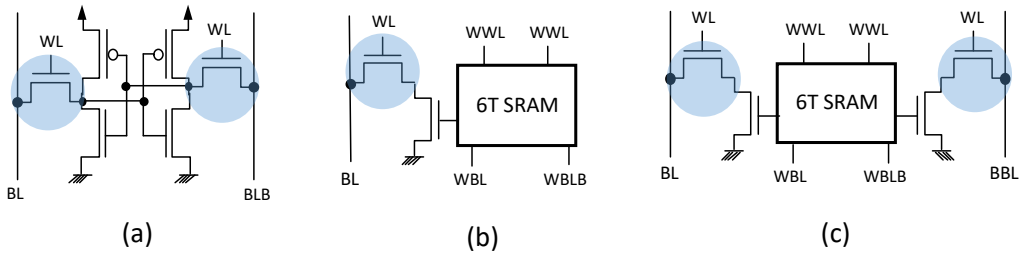


Figure 3.5: SRAM bit-cells employed for in-memory computing: (a) standard 6T [28], (b) single-ended 8T [34], and (c) a 10T [29]. Access transistors that contribute to current variations for each bit-cell are shaded.

below:

$$y = \left(\sum_{j=1}^{N} w_j x_j \right)_{\mathrm{a}} \qquad \text{(MM)} \qquad (3.33)$$

$$= \sum_{i=1}^{B_w} 2^{-i} \left(\hat{\mathbf{w}}_i^{\mathsf{T}} \mathbf{x} \right)_{\mathrm{a}} \qquad \text{(BM)} \qquad (3.34)$$

$$= \sum_{k=1}^{B_x} 2^{-k} \left(\mathbf{w}^{\mathsf{T}} \hat{\mathbf{x}}_k \right)_{\mathrm{a}} \qquad \text{(MB)} \qquad (3.35)$$

$$= \sum_{k=1}^{B_x} \sum_{i=1}^{B_w} 2^{-i-k} \left(\hat{\mathbf{w}}_i^{\mathsf{T}} \hat{\mathbf{x}}_k \right)_{\mathrm{a}} \qquad \text{(BB)} \qquad (3.36)$$

where, $(z)_{\mathrm{a}}$ indicates that the computations z are implemented in analog via one or more of the in-memory compute models QA, QS, and IA described in Section 3.2. The notation for the decomposition in (3.33)-to-(3.36) is obtained by concatenating symbols M (for multi-bit) and B (for binary) representing the precision of weights and activation in order, e.g., BM indicates that the weights are binary and the activations are multi-bit.

Though (3.33) to (3.36) assume unsigned $\mathbf{w}$ and $\mathbf{x}$, a similar set of expressions can be obtained for other number representations [35]. In this chapter, we limit our study to CMOS based in-memory architectures that accommodate signed weights $\mathbf{w}$ and unsigned inputs $\mathbf{x}$.

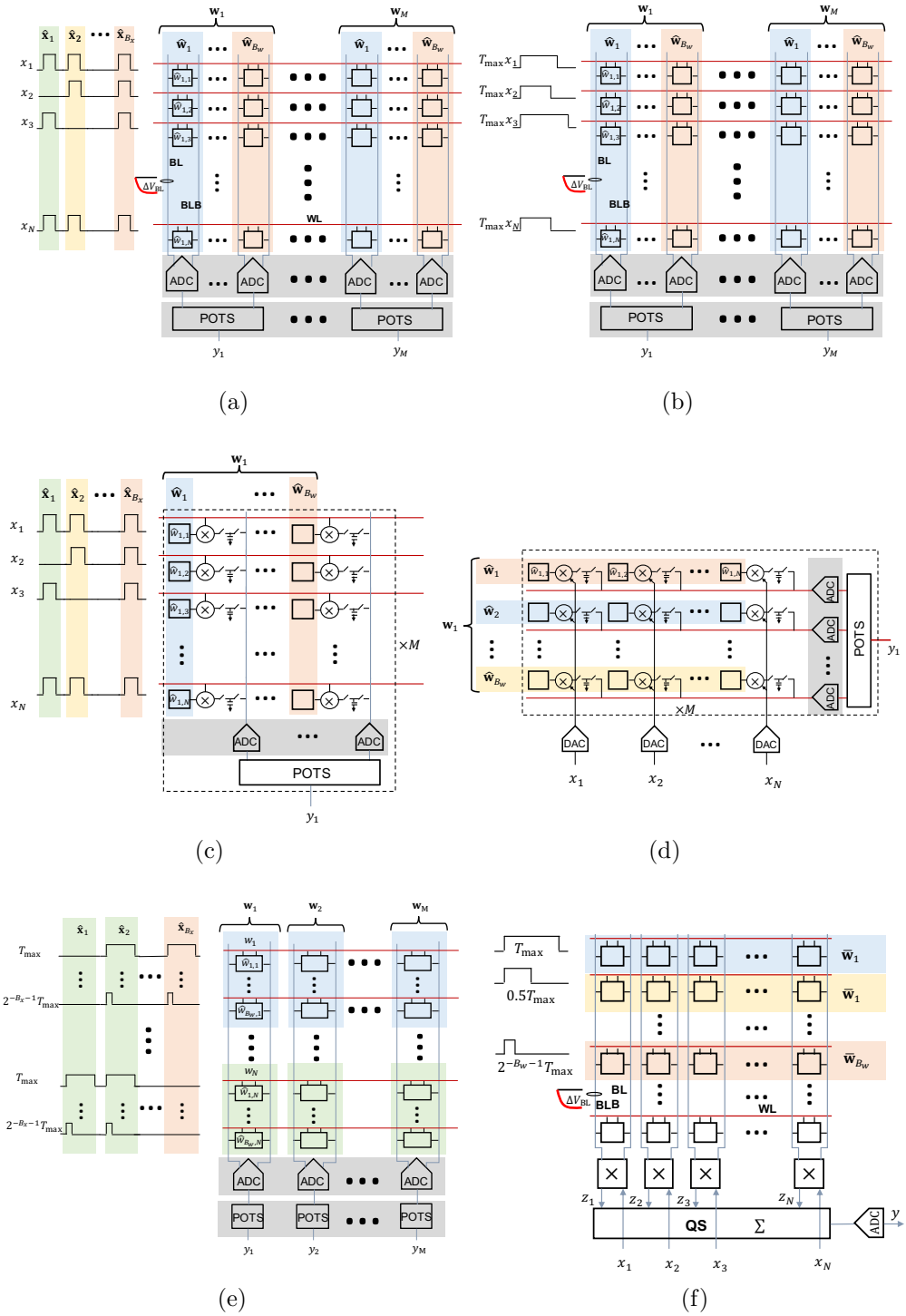


Figure 3.6: Systematic composition of in-memory architectures using the framework in Fig. 3.4: the (a) QA-BB, (b) QA-BM, (c) QS-BB, (d) QS-BM, (e) QA-BM, and the (e) CM architectures.

3.3.2 The QA-BB Architecture

The QA-BB architecture in Fig. 3.6(a) employs a 6T [38] or 8T [34] (see Fig. 3.5(a)-(c)) within the QA model (see Section 3.2.1) and the BB mixed-signal arithmetic decomposition method (3.36). This architecture implements a fully-binarized dot-products on the BLs by mapping the input bit $\hat{x}_{i,j}$ to the WL access pulse $V_{\rm WL}$ while the weights $\hat{w}_{i,j}$ are stored across B_w columns of the BCA so that the BC currents $I_{i,j} \propto \hat{w}_{i,j}$. The voltage discharge on the BL ($\Delta V_{\rm BL}$) and the BL capacitance $C_{\rm BL}$ correspond to $V_{\rm o}$ and C in (3.11), respectively. The QA-BB architecture sequentially (bit-serially) processes one multi-bit input vector $\mathbf{x}$ in B_x in-memory compute cycles followed by a power-of-two summing (POTS) of the binarized dot-products digitally to obtain the final multi-bit dot-product (3.1).

By substituting $C = C_{\rm BL}$ and $V_{\rm o} = \Delta V_{\rm BL}$ in (3.17), the average energy per dot-product $E_{\rm QA\text{-}BB}$ for the QA-BB architecture, assuming the use of the single-ended 8T-SRAM cell in Fig 3.5(b), is obtained as follows:

$$E_{\rm QA\text{-}BB} = B_w B_x E_{\rm QA} + B_w B_x E_{\rm adc} + E_{\rm misc} \tag{3.37}$$

where $E_{\rm adc}$ is the energy consumption of one ADC operation, and $E_{\rm misc}$ encapsulates the energy cost of pulse generation, driving switches, digital computations, charging intermediate capacitance, and others. Typically the first term in (3.37) dominates the overall energy consumption as $E_{\rm ADC}$ gets amortized over dot-product length N and $E_{\rm misc}$ is expected to be negligible.

3.3.3 The QA-BM Architecture

The QA-BM architecture in Fig. 3.6(b) employs a 6T [96] or 8T SRAM BC [30] within the QA model (see Section 3.2.1) and the BM mixed-signal arithmetic decomposition method (3.34). This architecture implements binary-weighted dot-products on the BLs by mapping multi-bit inputs x_j to WL access pulse widths T_j while the weights $\hat{w}_{i,j}$ are stored across B_w columns of the BCA as in the QA-BB architecture. The QA-BM architecture processes one multi-bit input vector $\mathbf{x}$ in one in-memory compute cycle so that the voltage discharge $\Delta V_{\rm BL}$ on the BLs are proportional to binary-weighted dot-products. The column outputs are powers-of-two summed (POTS) to

obtain the final multi-bit dot-product (3.1).

The energy model of the QA-BM architecture is obtained by substituting $C = C_{\mathrm{BL}}$ and $V_{\mathrm{o}} = \Delta V_{\mathrm{BL}}$ in (3.17), the average energy per dot-product $E_{\text{QA-BM}}$ for the QA-BM architecture, assuming the use of the single-ended 8T-SRAM cell in Fig 3.5(b), is given by:

$$E_{\text{QA-BM}} = B_w E_{\mathrm{QA}} + B_w E_{\mathrm{adc}} + E_{\mathrm{misc}} \tag{3.38}$$

where the first term dominates the overall energy consumption as E_{adc} gets amortized over N and E_{misc} is small.

The throughput of the QA-BM architecture is higher as compared to the QA-BB architecture since its computations are completed in a single in-memory compute cycle. However, the energy-efficiency of QA-BB architecture scales better (linearly) with the input precision B_x (see (3.37)) compared to the QA-BM architecture whose energy scales exponentially with B_x (see (3.38) and (3.17)).

3.3.4 The QS-BB Architecture

The QS-BB architecture in Fig. 3.6(c) modifies the 6T BC to include a capacitor C_{o}, e.g., a MOM capacitor in [31, 35], and additional switches to realize multiplication within the bitcell and BB mixed-signal arithmetic decomposition method (3.36). This architecture implements a fully binarized dot-product by storing the weights $\hat{w}_{i,j}$ across B_w columns of the BCA and by mapping the binary input $\hat{x}_{i,j}$ to the multipliers. The multiplication operation proceeds by charging the capacitor C_{o} to V_{dd} based on the value of $\hat{x}_{i,j}$ and then discharging it based on $\hat{w}_{i,k}$. The multiplication operation is followed by a QS operation so that the voltage across the capacitors in each column is proportional to fully binarized dot-products. The QS-BB architecture sequentially (bit-serially) processes one multi-bit input vector $\mathbf{x}$ in B_x in-memory compute cycles followed by power-of-two summing of (POTS) the binarized dot-products digitally to obtain the final multi-bit dot-product (3.1).

The energy model of the QS-BB architecture is derived from (3.29) by substituting all C_j's with C_{o}. The average energy per dot-product $E_{\text{QS-BB}}$

for QS-BB architecture is given by:

$$E_{\text{QS-BB}} = B_x B_w (E_{\text{QS}} + N E_{\text{mult}}) + B_x B_w E_{\text{adc}} + E_{\text{misc}} \tag{3.39}$$
$$E_{\text{mult}} = \mathbb{E}[\hat{x}_{i,j}(1 - \hat{w}_{i,j})] C_{\text{o}} V_{\text{dd}}$$

where E_{mult} is the average energy required for the switching operations that implement a multiplication. Typically, the first term in (3.39) dominates the overall energy consumption as E_{adc} gets amortized over the dimension N and E_{misc} is small.

3.3.5 The QS-BM Architecture

The QS-BM architecture [29] in Fig. 3.6(d) employs a modified BC to include a capacitor C_{o} and additional switches for multiplication within the QS model (see Section 3.2.3) and BM mixed-signal arithmetic decomposition method (3.34). While the works such as [29] employ the parasitic capacitances on the BL within the BC, an explicit MOM capacitor is assumed in this analysis for simplicity. This architecture implements a binary weighted dot-product by storing the weights $\hat{w}_{i,j}$ across B_w rows of the BCA and by providing multi-bit analog input x_j to the multiplier. The multiplication is implemented by first charging the capacitor C_{o} to a voltage proportional to x_j and then discharging it based on $\hat{w}_{i,k}$. The multiplication is followed by a QS operation across the rows so that the final voltage across the capacitors in each row is proportional to binary-weighted dot-product. Thus, the QS-BM architecture processes one multi-bit input vector $\mathbf{x}$ in one in-memory compute cycle to compute binary-weighted dot-product that are power-of-two summed (POTS) digitally to obtain the final multi-bit dot-product (3.1).

The average energy per dot-product $E_{\text{QS-BM}}$ for the QS-BB architecture is obtained from (3.29) as:

$$E_{\text{QS-BM}} = B_w (E_{\text{QS}} + N E_{\text{mult}} + E_{\text{adc}}) + E_{\text{misc}}$$
$$E_{\text{mult}} = \mathbb{E}[x_j(1 - \hat{w}_{i,j})] C_{\text{o}} V_{\text{dd}} \tag{3.40}$$

where E_{misc} also includes the energy consumption of the digital to analog converters (DACs) used for converting x_j into the analog domain. Since the DACs are shared across multiple dot-products, its energy is expected to get

amortized over the number of rows. QS-BM achieves a higher throughput than QS-BB architecture since it executes all its computations in one in-memory compute cycle. QS-BB and QS-BM have comparable energy for the same inference accuracy as seen from (3.39)-(3.40).

3.3.6 The QA-MB Architecture

The compositional framework (see Fig. 3.4) suggests the QA-MB architecture in Fig. 3.6(e) as a viable option even though it is yet to be implemented. The QA-MB architecture in Fig. 3.6(e) can employ a 6T or 8T (see Fig. 3.5(a)-(c)) within the QA model (see Section 3.2.1) and the MB mixed-signal arithmetic decomposition method (3.36). This architecture implements binary-input multi-bit weight dot-products on the BLs by storing w_j in a single column across NB_w rows of the BCA, while mapping its bits to powers-of-two weighed WL access pulse-widths T_j and the input bits $\hat{x}_{i,j}$ to the WL access pulse amplitude $V_{\rm WL}$. The voltage discharge on the BL ($\Delta V_{\rm BL}$) and the BL capacitance $C_{\rm BL}$ correspond to $V_{\rm o}$ and C in (3.11), respectively. The QA-MB architecture sequentially (bit-serially) processes one multi-bit input vector $\mathbf{x}$ in B_x in-memory compute cycles that result the binary-input multi-bit weight dot-products per-column. This is followed by a power-of-two summing digitally to obtain the final multi-bit dot-product (3.1).

By substituting $C = C_{\rm BL}$ and $V_{\rm o} = \Delta V_{\rm BL}$ in (3.17), the average energy per dot-product $E_{\text{QA-MB}}$ for the QA-BB architecture, assuming the use of the single-ended 8T-SRAM cell in Fig 3.5(b), is obtained as follows:

$$E_{\text{QA-MB}} = B_x E_{\rm QA} + B_x E_{\rm adc} + E_{\rm misc} \tag{3.41}$$

where $E_{\rm adc}$ is the energy consumption of one ADC operation, and $E_{\rm misc}$ encapsulates the energy cost of pulse generation, driving switches, digital computations, charging intermediate capacitance, and others. Typically the first term in (3.41) dominates the overall energy consumption as $E_{\rm ADC}$ gets amortized over the dot-product length N and $E_{\rm misc}$ is expected to be negligible. QS-MB and QS-BB have comparable throughput. However, for the same accuracy, QA-MB is more energy efficient especially with increasing B_w, as seen from (3.39)-(3.41).

3.3.7 Compute Memory (CM)

CM [26, 28, 49] in Fig. 3.6(f) employs a 6T SRAM BC within the QA (see Section 3.2.1) and QS (see Section 3.2.3) models and the MM mixed-signal arithmetic decomposition (3.33). In the most general case, CM strives to implement a $B_w \times B_x$-b dot product directly by mapping B_x-b inputs x_j to pulse width T_j and/or amplitude $V_{\mathrm{WL},j}$ of the WL access pulses and storing B_w-b weights w_j in a column-major format across $B_w \times N$ rows. In practice, CM realizations such as [28] employ powers-of-two weighted WL access pulse-widths for B_w rows so that the voltage discharge ΔV_{BL} on the j-th BL is proportional to the weight w_j. The product $w_j x_j$ is realized using a per-column charge redistribution-based multiplier, followed by a QS stage to aggregate the N multiplier outputs. In this way, CM computes the $B_w \times B_x$-b dot-product (3.1) in analog in a single in-memory compute cycle.

The average energy per dot-product E_{CM} for CM is obtained by substituting $C = C_{\mathrm{BL}}$ and $V_{\mathrm{o}} = \Delta V_{\mathrm{BL}}$ in (3.17), and using (3.29) to get:

$$E_{\mathrm{CM}} = 2NE_{\mathrm{QA}} + E_{\mathrm{QS}} + E_{\mathrm{mult}} + E_{\mathrm{adc}} + E_{\mathrm{misc}} \tag{3.42}$$

where E_{mult} is the energy consumption of the mixed-signal multiplier. The first term in (3.42) is the energy consumed by the QA stage where the factor 2 accounts for discharge on both BL and BL-bar required to realize signed weights [49] using a standard 6T-SRAM. The second term (3.42) is the energy consumed by the aggregation due to the use of the QS model with identical capacitors C_{o}.

CM's unique feature is its ability to realize multi-bit product $w_j x_j$ on the BLs in a single in-memory compute cycle. While binary-weighted architectures QA-BB, QA-BM, QS-BB and QS-BM consume the BL dynamic range to realize binary-weighted dot-products, CM does so to implement a few (single) multi-bit products. Similar to QA-BB and QS-BB which employ bit-serial inputs, CM needs multiple in-memory compute cycles to realize a multi-bit MVM. The number of cycles to realize a multi-bit MVM in CM scales with matrix dimensionality while in QA-BB and QS-BB, the number of cycles is proportional to the input precision.

3.4 Conclusions

The chapter presents a compositional framework to understand the relationship between various in-memory architectures. The existing in-memory architectures can be derived using this framework, and are tabulated in Table 3.1. Many new architectures that can be derived from this framework. It is not clear how these architectures compare with each other, and when one architecture should be chosen over the other in terms of latency, energy and accuracy trade-offs. This trade-off is difficult to quantify even for a specific architecture because it is a function of precision, dimensionality and the compute model being used. Chapter 4 presents a methodology to analyze in-memory architectures in terms of its compute SNR and energy consumption.

Derivations of Equations (3.13), and (3.15)

Derivation of (3.13):
We first use an α-law transistor current equation for modeling the SRAM access transistor current I_j (see Fig. 3.3(a)):

$$I_j = \frac{W}{L} k'(V_{\rm WL} - V_{\rm t})^{\alpha} \tag{3.43}$$

Assuming that the threshold voltages are randomly distributed with a mean $V_{\rm t}$ and standard deviation $\sigma_{V_{\rm t}}$, variations in the cell currents (i_j) due to threshold voltage variations ($\delta V_{\rm t}$) can be approximated using a first order Taylor expansion:

$$\begin{aligned} I_j + i_j &\approx I_j + \delta V_{\rm t} \frac{\partial I_j}{\partial V_{\rm t}} \\ &= \frac{W}{L} k'(V_{\rm WL} - V_{\rm t})^{\alpha} - \delta V_{\rm t} \big(\alpha \frac{W}{L} k'(V_{\rm WL} - V_{\rm t})^{\alpha-1}\big) \\ &= I_j - \delta V_{\rm t} \frac{\alpha I_j}{V_{\rm WL} - V_{\rm t}} \end{aligned} \tag{3.44}$$

Thus, the current mismatch i_j has zero mean, and standard deviation σ_{I_j} as captured in (3.13).

Derivation of (3.15):

A realistic V_{WL} curve would have a non-linear rise/fall curves (black curve in Fig. 3.3(b)). We use a simple linear approximation can be made (red curve in Fig. 3.3(b)) in order to simplify the analysis. Under this linear approximation, and modeling the SRAM cell current as an ideal current source with α-law current equation (3.43), the total current discharge associated with j-th cell $V_{\mathrm{o},j}$ can be shown to be:

$$V_{\mathrm{o},j} + v = \frac{I_j}{C}\Big[T_j - T_{\mathrm{r}} + \Big(\frac{V_{\mathrm{WL}} - V_{\mathrm{t}}}{V_{\mathrm{WL}}}\Big)\frac{T_{\mathrm{r}} + T_{\mathrm{f}}}{\alpha + 1}\Big] \tag{3.45}$$

If the V_{WL} pulse was ideal, then $T_r = T_f = 0$ and the expression reduces to $\Delta V_{\mathrm{o},j} = \frac{I_j T_j}{C_{\mathrm{BL}}}$, the same expression as before for a single cell activated for duration T_j.

$$V_{\mathrm{o},j} = \frac{I_j}{C}(T_j - t_{\mathrm{rf}}) \tag{3.46}$$

Therefore, a correction factor t_{rf} described in (3.15) will account for the correction to the discharge time due to rise and fall effects of the WL pulse. This correction factor due to rise and fall time is predictable and can be largely corrected for by pre-calibrating the pulses.

CHAPTER 4

SIGNAL-TO-NOISE RATIO ANALYSIS OF IN-MEMORY ARCHITECTURES

Chapter 3 demonstrated how a specific choice of an in-memory compute model combined with a specific mixed-signal arithmetic decomposition method results in unique in-memory architecture, e.g., QA-BB, QA-BM, QS-BB, QS-BM, QA-MB, and CM. Each of these architectures has a unique compute SNR (Section 3.1) that describes the accuracy of their in-memory dot product computation. The compute SNR is dictated by the noise sources in the associated in-memory compute models modulated by the arithmetic decomposition employed by these architectures.

This chapter adopts a communication-inspired view of DIMA to analyze the compute SNR of DIMA and relate it to its energy consumption. The SNR analysis leverages the compositional framework, the compute models, and their associated noise models presented in Chapter 3. The SNR analysis presented this chapter allows us to predict in in-memory architecture trends and develop design guidelines for in-memory IC designers.

4.1 SNR Analysis of In-memory Architectures

This section derives expressions for the compute SNR for three distinct in-memory architectures – QA-BB in Fig. 3.6(a) (based on [30]), QS-BM architecture in Fig. 3.6(b) (based on [29, 31]), and CM in Fig. 3.6(c) (based on [28, 49]) – as representatives of the three compute models.

We use the compute SNR metrics – analog $\mathrm{SNR_a}$, digitization $\mathrm{SNR_d}$, and total $\mathrm{SNR_T}$ described in Section 3.1 – to characterize these architectures. $\mathrm{SNR_a}$ quantifies the impact of circuit non-idealities and input quantization noise, $\mathrm{SNR_d}$ quantifies the impact of digitization via the ADC, and $\mathrm{SNR_T}$ considers all the noise sources in the in-memory architectures. In this chapter, we will develop analytical expressions for these compute SNR metrics.

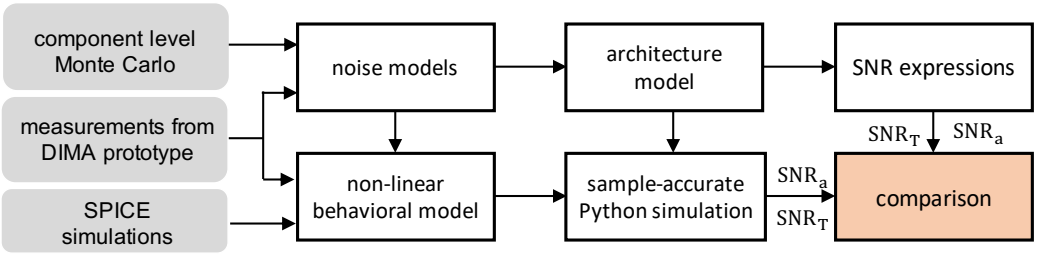


Figure 4.1: Methodology for validating the analog SNR ($\mathrm{SNR_a}$) and total SNR ($\mathrm{SNR_T}$) expressions for various architectures.

Such SNR analysis of in-memory architecture will enable circuit designers to evaluate in-memory architectures across its vast design space without having to perform computationally intensive sample-accurate simulations. However, such sample-accurate simulations would be necessary to validate the accuracy of the SNR analysis. Next, we present a validation methodology to do just that.

4.1.1 Validation Methodology

Figure 4.1 describes our compute SNR validation methodology. We obtain parameters for the noise models in Section 3.2 from component level Monte-Carlo simulations in TSMC 65 nm GP process. The noise model parameters were validated with measurements from our in-memory computing prototype IC [28, 53] when possible. The final noise model parameters are summarized in Table 3.2.

Empirical behavioral models incorporating the circuit non-linearity were developed from SPICE simulations. This is particularly important for the QA compute model, where the effects of non-linearity could impact SNR. These non-ideal analog effects and noise models were then incorporated to estimate the $\mathrm{SNR_a}$ for compute models. The average difference in SNR due to non-linearity was found to be 0.4 dB and 0.1 dB for QA and QS compute models, respectively.

Architectural SNR expressions were validated against the sample-accurate SNR of the in-memory architectures with noise injection using the noise models in Section 3.2. Noise injection was performed via instantiation of random variables with statistics as per noise models and reported results correspond to ensemble averages over 1000 instances.

The analysis in the following sections employs circuit and architecture parameter values listed in Table 3.2. We assume SRAM with the number of rows fixed at 512 for CM and QA-BB architecture resulting $C_{\rm BL} = 270\,{\rm fF}$ (in 65 nm CMOS). We assume the impact of rise and fall time γ_{rf} (3.15) is minimized via calibration. The energy and accuracy are tuned using $V_{\rm WL}$ in CM and QA-BB, and using $C_{\rm o}$ in QS-BM architectures. We assume zero-mean signed weights w_j drawn independently from a weight distribution and unsigned inputs x_j also drawn independently from an input distribution. Next, we begin with CM as it is an MM architecture employing both QA and QS models, thereby allowing us to explore trade-offs between various noise sources.

4.1.2 CM SNR Analysis

The CM architecture that implements signed operations is described in [49]. The expressions $\sigma^2_{\eta_{\rm c}}$ and $\sigma^2_{\eta_{\rm e}}$ for CM, derived at the end of this chapter, are as follows:

$$\sigma^2_{\eta_{\rm c}} = \frac{1}{12} N \mathbb{E}\left[x^2\right] \sigma_w^2 k_{\rm clip}^{-2} 2^{2B_w} \left(1 - 2k_{\rm clip} 2^{-B_w}\right)_+^2 \tag{4.1}$$

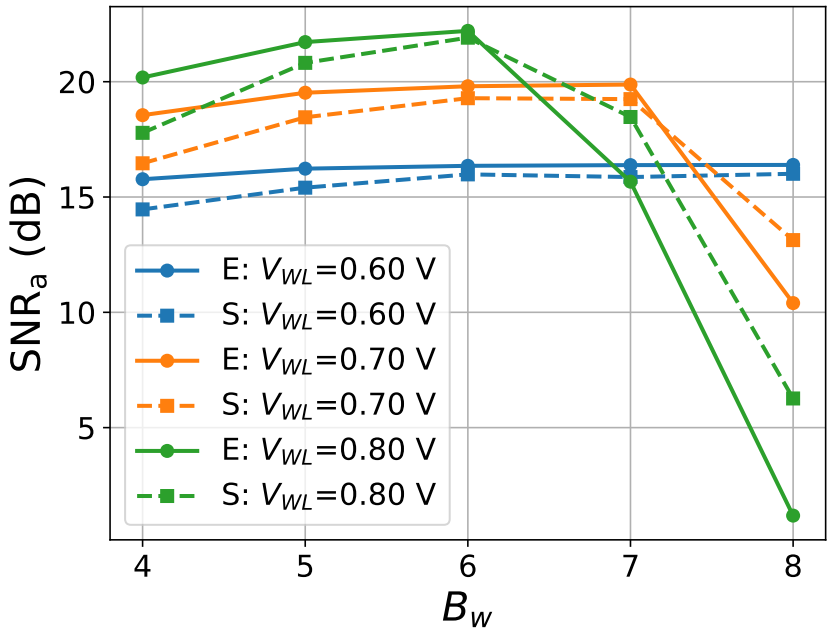
$$\sigma^2_{\eta_{\rm e}} = \frac{2}{3} N \mathbb{E}\left[x^2\right] \left(\frac{1}{4} - 4^{-B_w}\right) \sigma_{\rm D}^2 \tag{4.2}$$

where $(x)_+ = \max(x, 0)$, $k_{\rm clip}$ is the ratio of the maximal-to-unit BL discharge, and $\sigma_{\rm D}$ is the normalized standard deviation of bit-cell current mismatch obtained from (3.13), as follows:

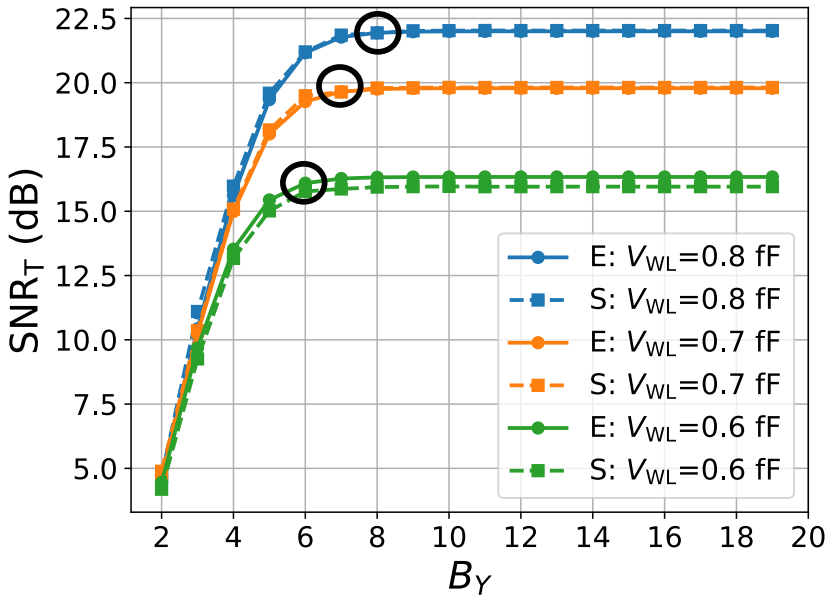
$$\sigma_{\rm D} = \frac{\sigma_{Ij}}{I_j} = \frac{\alpha \sigma_{V_{\rm t}}}{V_{\rm WL} - V_{\rm t}} \tag{4.3}$$

Since the variations in bit-cell currents dominate, (4.2) neglects the impact of pulse width variations, and the noise sources in QS. Note that, however, that these noise sources were considered while performing sample-accurate simulations used for comparison.

By substituting (3.10), (4.1), and (4.2) in (3.6) we obtain the $\rm SNR_a$ ex-



(a)



(b)

Figure 4.2: SNR trade-offs in CM with $B_x = 6$ and $N = 128$: (a) $\mathrm{SNR_a}$ as a function of B_w showing that an optimal value of B_w that balances quantization and clipping noise exists, and (b) $\mathrm{SNR_T}$ as a function of B_y with $B_w = 6$. 'E' denotes SNR obtained from (4.4) and 'S' denotes sample-accurate simulations using (3.12) and (3.25).

pression for CM as follows:

$$\mathrm{SNR_a} = \frac{12}{\frac{\Delta_w^2}{\sigma_w^2} + \frac{\Delta_x^2}{\mathbb{E}[x^2]} + k_{\mathrm{clip}}^{-2} 4^{B_w} \left(1 - 2k_{\mathrm{clip}} 2^{-B_w}\right)_+^2 + \frac{8\left(\frac{1}{4} - 4^{-B_w}\right)\sigma_{\mathrm{D}}^2}{\sigma_w^2}} \tag{4.4}$$

In (4.4), observe that the denominator contains a quantization noise term (first term) that decreases with B_w and a clipping noise term (third term) that increases with B_w. This implies that there exists an optimal value of B_w which maximizes $\mathrm{SNR_a}$. This can be seen in Fig 4.2(a) where the $\mathrm{SNR_a}$ peaks at $B_w = 6$ and $B_w = 7$ at $V_{\mathrm{WL}} = 0.8\,\mathrm{V}$ and $V_{\mathrm{WL}} = 0.7\,\mathrm{V}$, respectively.

Another interesting trade-off is between clipping and circuit noise. For example, for $B_w = 7$ in CM (Fig. 4.2(a)), $\mathrm{SNR_a}$ is dominated by circuit non-idealities when $V_{\mathrm{WL}} = 0.6\,\mathrm{V}$ and dominated by clipping noise when $V_{\mathrm{WL}} = 0.8\,\mathrm{V}$. Furthermore, the clipping noise and noise due to circuit non-idealities are balanced when operating with $V_{\mathrm{WL}} = 0.7\,\mathrm{V}$. In fact, we can show that the clipping threshold k_{clip} is proportional to $\sigma_{\mathrm{D}}^{\alpha}$ indicating this relationship.

In order to determine the output precision B_y, recall that:

$$\mathrm{SNR_d}\ (\mathrm{dB}) = \mathrm{SQNR}_y = 6B_y + 4.78 - \mathrm{PAR(dB)}, \tag{4.5}$$

where SQNR_y and PAR are the signal-to-quantization-ratio and peak-to-average-ratio of y_{o} in decibels, respectively.

In fact the PAR can be adjusted at the risk of clipping y_{o} before quantization. We need to choose the smallest $\mathrm{PAR} = \frac{\zeta\sigma_y}{\sigma_y} = \zeta$, while avoiding additional clipping noise at the output. We assume distribution of dot product output y_{o} to be Gaussian by virtue of the Central Limit Theorem. Under this assumption, if $\zeta \geq 4$ then $\geq 99.7\%$ outputs avoid clipping. This is empirically observed to yield minimal loss in $\mathrm{SNR_T}$ as shown in Fig 4.3. Therefore, substituting $\mathrm{PAR} = 12\,\mathrm{dB}$ in (4.5) we obtain $\mathrm{SNR_d(dB)} = 6B_y - 7.2\,\mathrm{dB}$. Furthermore, recall that we need $\mathrm{SNR_d} > \mathrm{SNR_a} + 9\,\mathrm{dB}$ to keep $\mathrm{SNR_T}$ within half a dB of $\mathrm{SNR_a}$. Hence, we obtain the following lower bound on the output precision as:

$$B_y > \frac{\mathrm{SNR_a(dB)} + 16.2}{6} \tag{4.6}$$

Figure 4.2(b) shows that the choice of B_y using (4.6) and $\zeta = 4$ ensures that the $\mathrm{SNR_T}$ is indeed within $0.5\,\mathrm{dB}$ from the maximum. We also note

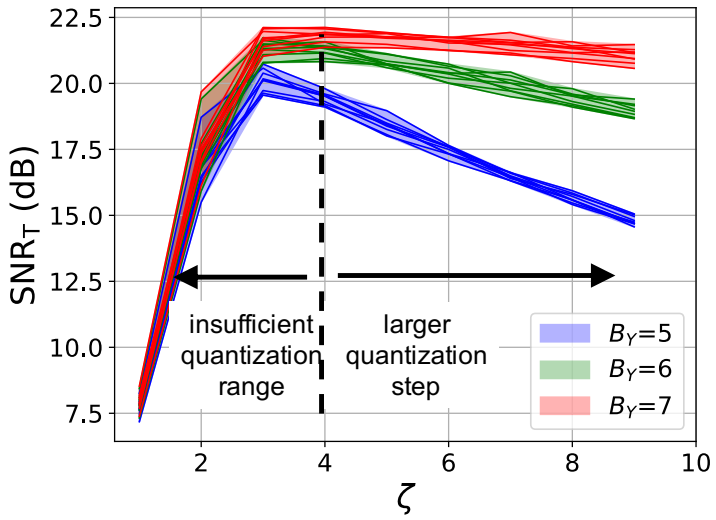


Figure 4.3: SNR_T of CM as a function of PAR (ζ) with $B_w = 6$, $B_x = 6$ and $V_\text{WL} = 0.8\,\text{V}$. Each plot corresponds to a value of N ranging from 50 to 500 highlighting the fact that SNR_T is independent of the dot product length.

that SNR_T is independent of the dot product dimensionality N as seen in Fig 4.3.

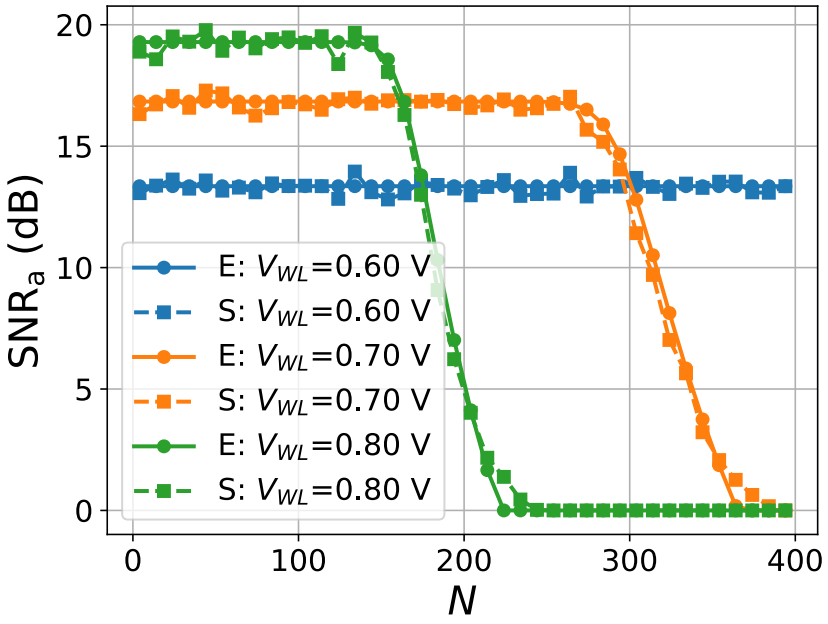
The ADC needs to quantize the output y in the range $[-\zeta\sigma_y, \zeta\sigma_y]$. In order to design the appropriate ADC for CM we need to find the voltage domain equivalent for this quantization range. The quantization range in the voltage domain (V_r) at the ADC input in CM (see Fig 3.6(e)) is given by:

$$V_\text{r} = \frac{4\sigma_w 2^{B_w} \Delta V_{\text{BL,LSB}} \sqrt{\mathbb{E}\left[x^2\right]}}{\sqrt{N}} \tag{4.7}$$

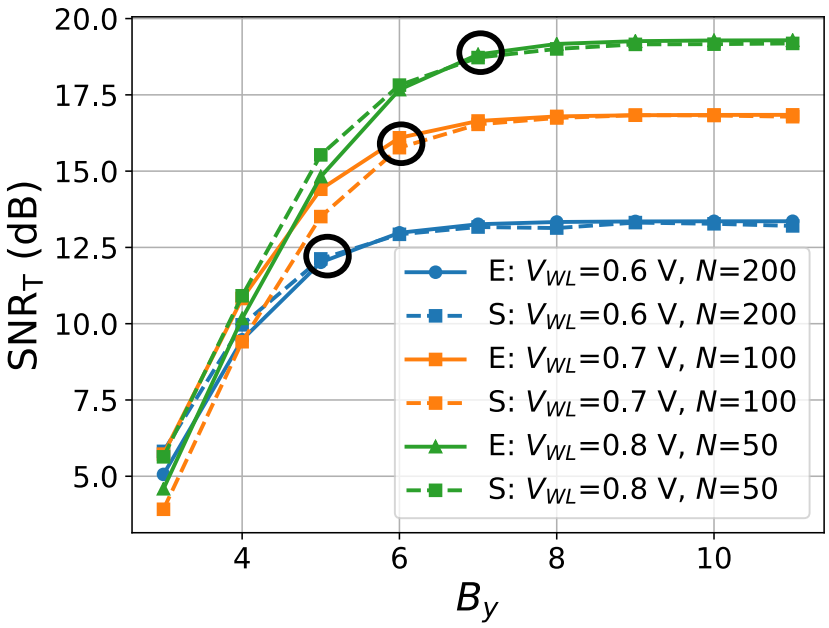
where $\Delta V_{\text{BL,LSB}}$ is the unit bitline discharge. In (4.7), the factor of $\sqrt{N}$ in the denominator is due to the use of QS to aggregate multiplier outputs in CM. The corresponding ADC quantization step (Δ_adc) in voltage domain is given by $V_\text{r}/2^{B_y}$.

4.1.3 QA-BB SNR Analysis

In the QA-BB architecture, there are $B_x \times B_w$ sources of clipping noise, each corresponding to the binarized dot products between $\{\hat{\mathbf{x}}_i\}_{i=1}^{B_x}$ and $\{\hat{\mathbf{w}}_i\}_{i=1}^{B_w}$. The clipping noise from each of the binarized dot products contributes to the combined clipping noise at the output at different magnitudes because of the



(a)



(b)

Figure 4.4: SNR trade-offs in the QA-BB architecture with $B_x = 6$ and $B_w = 6$: (a) $\mathrm{SNR_a}$ as a function of N for different values of V_{WL} showing that clipping noise causes the SNR to drop as sufficiently high values of N, and (b) $\mathrm{SNR_T}$ as a function of B_y showing that (4.10) correctly predicts the required B_y. 'E' denotes evaluation of the expressions (4.8) and (4.9) and 'S' denotes sample-accurate simulations of (3.12).

arithmetic in (3.36). The combined clipping noise reflected at the output y, derived at the end of this chapter, is given by:

$$\sigma_{\eta_c}^2 = \frac{4}{9}\left(1-4^{-B_w}\right)\left(1-4^{-B_x}\right)\sum_{k=k_{\text{clip}}}^{N}\left(k-k_{\text{clip}}\right)^2\binom{N}{k}\left(\frac{1}{4}\right)^k\left(\frac{3}{4}\right)^{N-k} \tag{4.8}$$

where k_{clip} is the ratio of the maximal-to-unit BL discharge.

Furthermore, each binarized dot product discharge also suffers from circuit noise, which can be aggregated to obtain the total circuit noise variance given by:

$$\sigma_{\eta_e}^2 = \frac{N\sigma_{\text{D}}^2\left(1-4^{-B_w}\right)\left(1-4^{-B_x}\right)}{9} \tag{4.9}$$

where σ_{D} is the normalized standard deviation bit-cell current mismatch (4.3).

In QA-BB architecture, quantization noise minimally trades off with the clipping noise and the noise due to circuit non-idealities. This is because the final dot product y is obtained by the aggregation of many independently computed binarized dot products. Nevertheless, we do observe a trade-off between clipping ($\sigma_{\eta_c}^2$) and noise due to circuit non-idealites ($\sigma_{\eta_e}^2$). Here, $\sigma_{\eta_c}^2$ decreases by reducing V_{WL} as k_{clip} increases. However reducing V_{WL} increases σ_{D} as well, thereby increasing $\sigma_{\eta_e}^2$. Since $\sigma_{\eta_c}^2$ limits N and $\sigma_{\eta_e}^2$ limits SNR_{a}, QA architectures exhibit a trade-off between dot product size N and SNR_{a} as seen in Fig. 4.4(a).

In QA-BB architecture, B_wB_x ADC operations are performed. We assume that each conversion utilizes the same precision B_y to convert intermediate binarized dot products. As this architecture corresponds to integer summing, the precision required is bounded by:

$$B_y > \log_2\left(\min\left(N, k_{\text{clip}}\right)\right) \tag{4.10}$$

and the corresponding quantization range in the voltage domain (V_{r}) at the input of the ADCs, derived at the end of the chapter, is given by:

$$V_{\text{r}} = \min(N, k_{\text{clip}})\Delta V_{\text{BL,LSB}} \tag{4.11}$$

where $\Delta V_{\text{BL,LSB}}$ is the unit discharge on the BL. This choice of output pre-

cision and dynamic range is validated in Fig. 4.4 (b).

4.1.4 QS-BM

Since clipping does not occur in the QS compute model, the primary source of noise is due to circuit non-idealities ($\sigma^2_{\eta_e}$). For QS-BM architecture $\sigma^2_{\eta_e}$, derived at the end of this chapter, is as follows:

$$\sigma^2_{\eta_e} = \frac{2}{3}(1 - 4^{-B_w})N \left(\frac{\mathbb{E}\left[x^2\right] \sigma^2_{C_o}}{C_o^2} + \frac{2\sigma^2_\theta}{V_{dd}^2} \right) + \sigma^2_{inj} \tag{4.12}$$

where the capacitor mismatch variance $\sigma^2_{C_o}$, and thermal noise variance σ^2_θ can be obtained using (3.26) and (3.28), respectively. Standard deviation of charge injection noise $\sigma_{Inj} = \sqrt{N\mathbb{E}\left[x^2\right]}\sigma_w WLC_{OX}/C_o$ is derived from (3.27), and we assume the mean of charge injection noise does not affect the SNR as it can be easily corrected for. The QS-BM architecture demonstrates a clear energy-accuracy-area trade-off as the SNR improves with larger C_o (see Fig. 4.5(a)) leading to a greater energy and area costs.

In QS-BM, B_w ADCs are invoked per dot product, and we assume they all have the same precision B_y. To obtain $\mathrm{SNR_d} > \mathrm{SNR_a} + 9\,\mathrm{dB}$, we need a precision of:

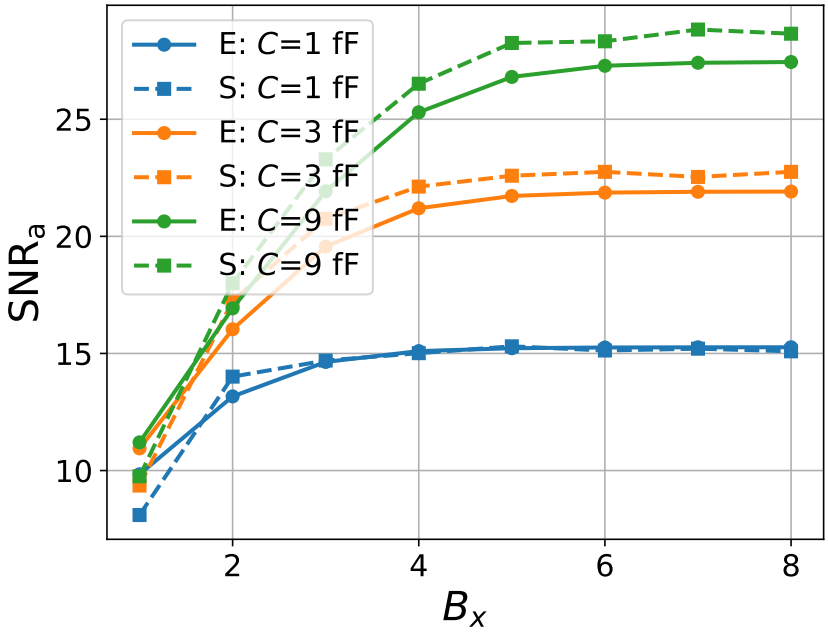
$$B_y > \frac{\mathrm{SNR_a(dB)} + 16.2}{6} \tag{4.13}$$

and the corresponding quantization range in the voltage domain (V_r) at the input of the ADCs, derived at the end of the chapter, is given by:

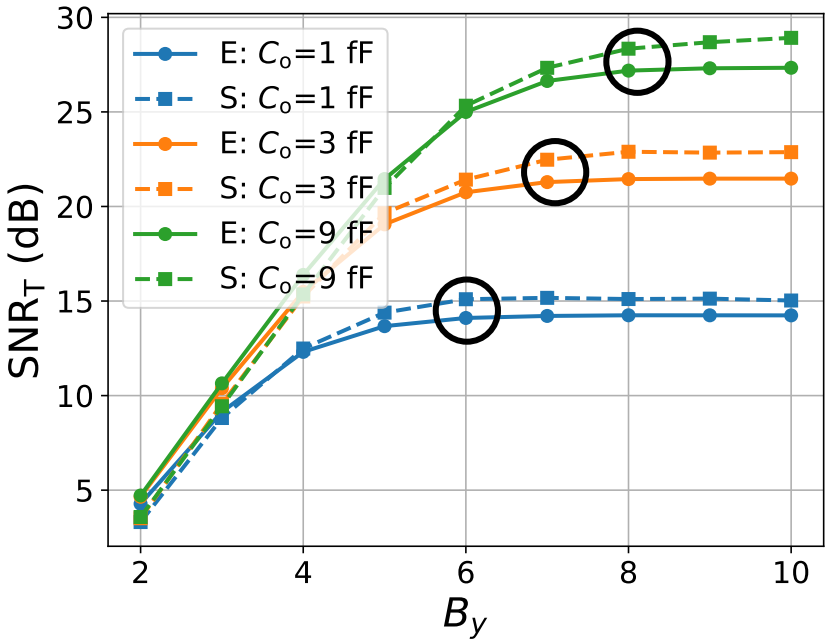
$$V_r = 4V_{dd}\sqrt{\frac{\mathbb{E}\left[x^2\right] + \sigma_x^2}{N}} \tag{4.14}$$

where the factor of $\sqrt{N}$ in the denominator stems is due to the charge sharing in QS compute model (3.24).

Figure 4.5(b) shows that predicted B_y results in a SNR loss of less than 0.5 dB as intended.



(a)



(b)

Figure 4.5: SNR trade-offs in the QS-BM architecture with $B_w = 7$, and $N = 64$: (a) $\mathrm{SNR_a}$ as a function of B_x for different values of C_o showing that the SNR improves with C_o, and (b) $\mathrm{SNR_T}$ as a function of B_y for different values of C_o with $B_x = 6$ and $B_w = 7$, showing that (4.13) correctly predicts the required B_y. Here, 'E' denotes evaluation of (4.12) and 'S' denotes sample-accurate simulations of (3.25).

4.2 Trends and Trade-offs

Since in-memory architectures exhibit a fundamental trade-off between their compute SNR and energy consumption, in this section, we study this trade-off as a function of the dimensionality N, input precision B_x, and weight precision B_w. We discuss the practical limitations associated with large dot product sizes. We show the impact of the choice of compute models and the dot product lengths on the ADC requirements. Furthermore, we provide guidelines for choosing and designing suitable architectures and corresponding operating conditions to maximize the energy-delay product (EDP) for a given required accuracy.

The trade-off between the compute SNR and energy is studied by leveraging the compute SNRs, and energy models for QA-BB (3.37), QS-BM (3.40), and CM (3.42) architectures. The parameters for these models are listed in Table 3.2.

4.2.1 Impact of SNR Requirements

From the energy models, it is clear that energy consumption decreases with decreasing SNR requirements. However, the rate at which the energy changes depends on the compute model employed by the architectures. We find that architectures dominated by QA noise such as CM and QA-BB are more sensitive to SNR requirements than the QS-BM architecture.

In the following, we focus on the energy consumption of the BCA only since it represents the bulk of the total energy consumption of in-memory architectures, i.e., we set $E_{\text{misc}} = E_{\text{adc}} = 0$ in (3.37), (3.40), and (3.42). Figure 4.6 shows that the BCA energy consumption for all three architectures reduces with the analog SNR SNR_a. In particular, we find that for every 6 dB drop in SNR_a, the compute energy reduces by 3.3× in CM and QA-BB, and by 2× in QS-BM. Furthermore, SNR_a of QA-BB suffers a catastrophic drop before it reaches the limit established by input quantization noise. This occurs due to an increase in the clipping noise variance $\sigma_{\eta_c}^2$, whereas QS-BM and CM are able to approach the quantization noise limit as clipping noise is small in the case of CM and clipping noise does not exist in QS-BM.

4.2.2 Impact of Weight and Input Precision

The impact of weight and input precision on the energy of in-memory computing depends on the way the multi-bit weights and inputs are mapped on these architectures. The energy of QA-BB architecture scales with both input and weight precision as it needs to compute $B_x B_w$ binarized dot products to arrive at the final output. On the other hand, the energy of QS-BM architecture depends only on B_w as only the weights are decomposed for dot product computations. Comparing Fig. 4.6(a) and Fig 4.6(b) shows the effects of changing the input precision B_x. The energy of QA-BB reduces while the energy of QS-BM and CM shows minimal change.

Weight precision affects all three architectures but in different proportions. CM shows the most change with B_w as the BL discharge $\Delta V_{\mathrm{BL}} \propto 2_w^B$. This can be observed by comparing the CM plot in Fig. 4.6(a) and Fig. 4.6(b).

4.2.3 Area-vs-SNR trade-offs

The energy and accuracy of the compute models are limited due to device mismatches. These mismatches can be potentially mitigated at the cost of area and hence memory density. For example, doubling the lengths and widths of the access transistors can reduce the threshold voltage mismatches by $2\times$ in QA, leading to a 6 dB gain in SNR with similar energy. Similarly, area trade-offs are seen in QS, where increasing capacitor size improves accuracy.

4.2.4 Trends with Technology Scaling

The energy and accuracy trade-offs strongly depend on the underlying process parameters. The simulations in Fig. 4.6 are based on 65 nm CMOS process, and are generated using parameters in Table 3.2. As the CMOS process technology scales, we expect improved energy efficiency and throughput due to lower capacitance and lower supply voltage. However, we need to also consider the impact of technology scaling on the noise sources. To study the impact of technology scaling, we employed the SNR models from Section 4.1 and energy models ((3.37), (3.42), and (3.40)) with parameters scaled as per the ITRS roadmap [97].

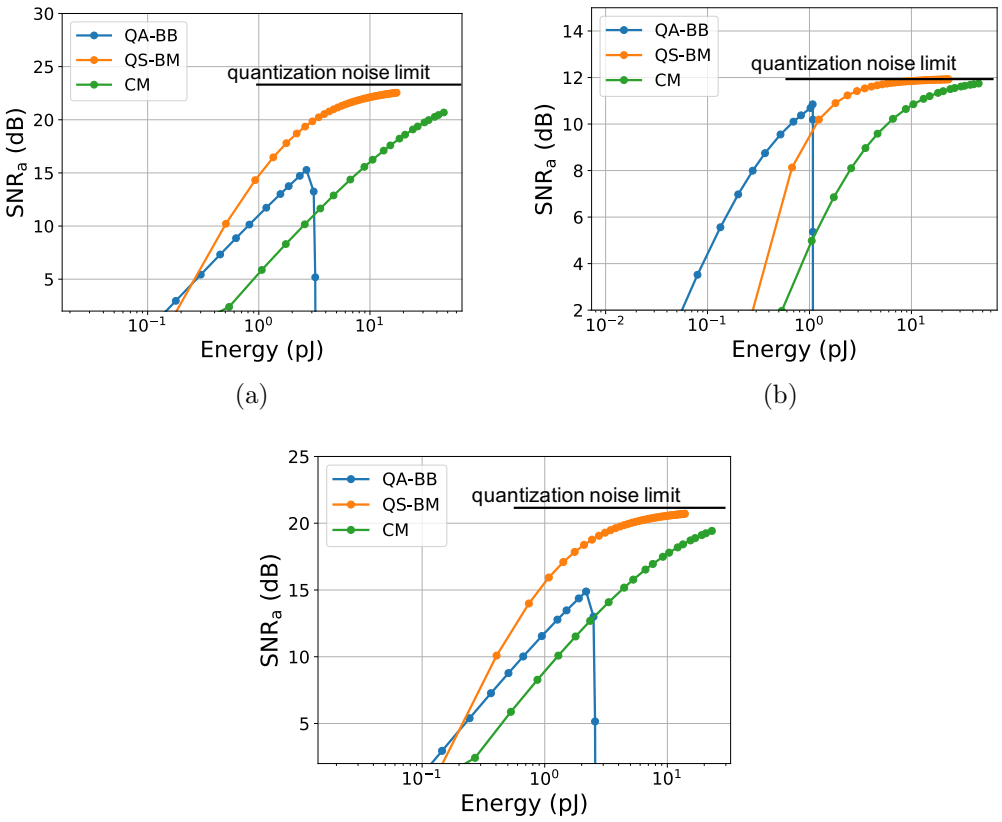


Figure 4.6: Energy consumption trends as a function of $\mathrm{SNR_a}$ with $N = 300$ for: (a) $B_x = 3$ and $B_w = 5$, (b) $B_x = 1$ and $B_w = 5$, and (c) $B_x = 3$ and $B_w = 4$. Energy traded-off by tuning V_{WL} in CM and QA-BB, and using C_{o} in QS-BM. The energy of consumption for all architectures increases linearly with N as indicated by (3.37), (3.40) and (3.42). Note that E_{adc} and E_{misc} are assumed to be negligible.

While in 65 nm CMOS process, the ratio of supply-to-threshold voltage ($V_{\mathrm{dd}}/V_{\mathrm{t}}$) is around 2.5, it tends to decrease with technology scaling. The lowered supply results in a higher probability of clipping for the same BL swing in QA based architectures. The increased clipping noise in advanced processes reduces the maximum achievable SNR in QA-BB architectures, as seen in Fig 4.7(b). Furthermore, increased WL resistance due to width-dependent scattering [98] in $\leq$ 16 nm technology nodes limits the smallest WL pulse-width T_o, thereby increasing the clipping probability and energy consumption. This effect can be observed in QA-BB (Fig 4.7(b)), and in CM (Fig 4.7(a)), where the energy consumption in 11 nm and 7 nm technology

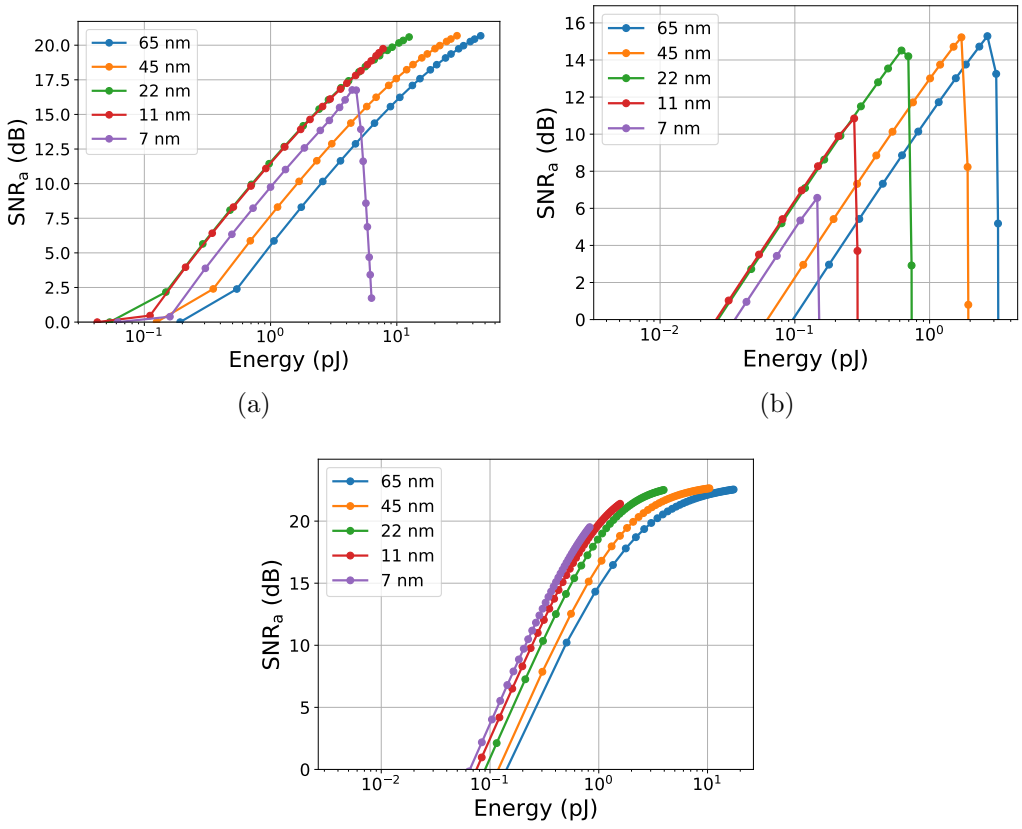


Figure 4.7: Impact of CMOS technology scaling on the energy-vs-SNR$_a$ trade-offs in: (a) CM, (b) QA-BB, and (c) QS-BM architectures with $B_x = 3$, $B_w = 5$, and $N = 300$. Technology scaling trends estimated based on ITRS tables [97]. FDSOI technology is assumed for 22 nm, 11 nm and 7 nm nodes.

nodes is higher than that at 22 nm for the same SNR. QS-BM architectures (Fig 4.7(c)), due to the absence of clipping noise, show an improvement in energy and throughput with technology scaling. However, due to the increased wire-load capacitance and resistance, throughput improvements slow down for 11 nm and 7 nm technology nodes.

4.2.5 Challenges in Analog-to-Digital Conversion

The cost of ADC becomes particularly important while operating with a small V_r, since high-resolution ADCs are required. ADCs that are pitch matched to the array are typically noise-limited as they operate with very limited area budget. For example, an ADC with 1 mV quantization step will

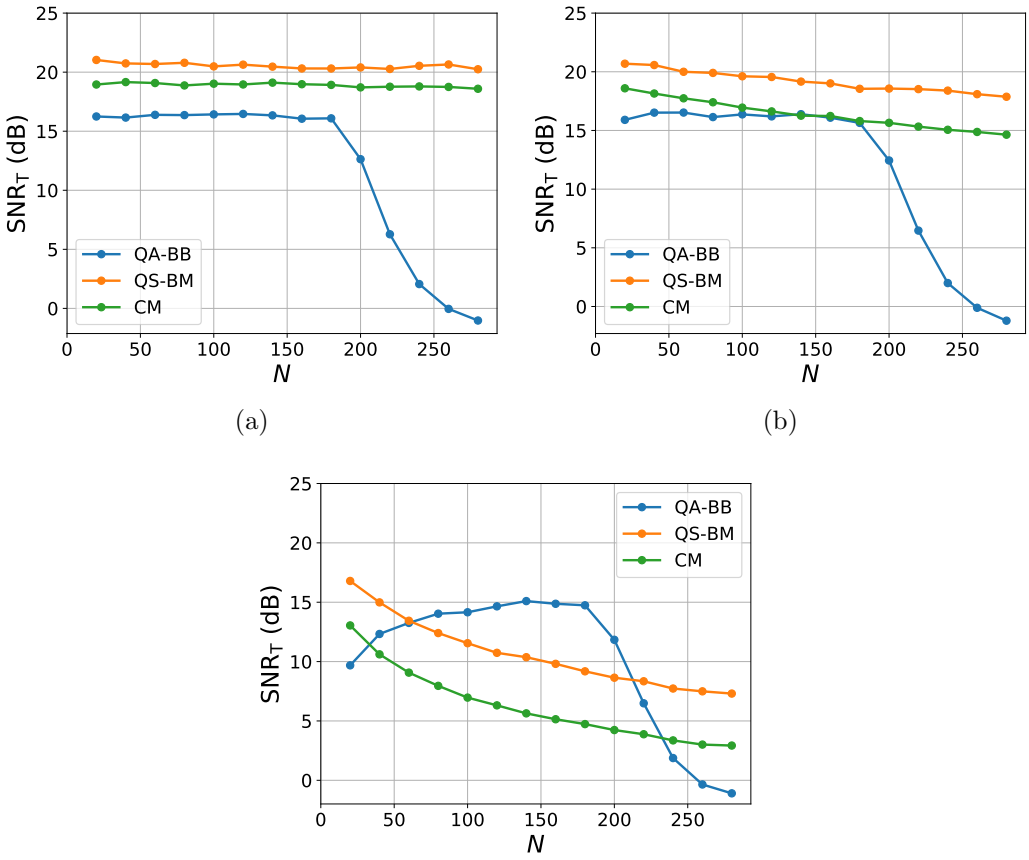


Figure 4.8: Impact of ADC quantization step on $\mathrm{SNR_T}$ with $B_x = 3$ and $B_w = 5$ as a function of N using ADCs with quantization steps of: (a) 1 mV, (b) 5 mV, and (c) 25 mV. CM and QA-BB operate with $V_{\mathrm{WL}} = 0.8\,\mathrm{V}$ and $V_{\mathrm{WL}} = 0.75\,\mathrm{V}$, respectively, and $C_{\mathrm{o}} = 4\,\mathrm{fF}$ in QS-BM.

require the thermal noise floor to be in the order of 0.3 mV to guarantee accurate conversion with >99% probability, thus requiring capacitors in the order of 100 fF. Such a capacitor needs the same area as 20 6T SRAM bit-cells. Energy costs of noise-limited ADC designs can be modeled [99, 100] as:

$$E_{\mathrm{adc}} = \frac{\beta}{\Delta_{\mathrm{adc}}^2} = \frac{\beta}{V_{\mathrm{r}}^2} 4^{By} \tag{4.15}$$

where Δ_{adc} is the ADC step size in voltage domain, V_{r} is the voltage range that need to be quantized, and β can be estimated from the Schreier figure of merit [101, 102]. Considering the recent ADCs tabulated in [103], the

best Schreier figure of merit is about 180 dB as of the year 2019, leading to $\beta = 7.5 \times 10^{-4}$ fJV^2 with $V_{\mathrm{dd}} = 1$ V.

The ADC energy can easily dominate while operating with large dot product lengths in architectures that use QS compute models. Equations (4.7) and (4.14) show that V_{r} reduces with increasing N in CM and QS-BM architectures, respectively.

Figure 4.8 shows $\mathrm{SNR_T}$ as a function of N with fixed quantization step Δ_{adc}. We find that operating with $\Delta_{\mathrm{adc}} = 1$ mV does not impact $\mathrm{SNR_T}$ in any architecture, as the required Δ_{adc} is greater than 1 mV for all N. Operating with $\Delta_{\mathrm{adc}} = 5$ mV results in $\mathrm{SNR_T}$ decreasing for QS-BM and CM, while QA-BB remains unaffected. Interestingly, operating with $\Delta_{\mathrm{adc}} = 25$ mV affects QA-BB only for $N < 140$. This is because V_{r} in QA-BB in fact increases with N. Note that designs with such increase in V_{r} may result in clipping for large N.

4.3 Design Guidelines

Discussion in Section 4.1 and 4.2 shows the relationships between the dot product length N, the SNR, computation energy, and ADC energy. It is clear that the application requirements determine the choice of architectures and that there is not a single architectural choice that is optimal for all scenarios. Below are a few considerations for designing in-memory architectures based on the applications requirements:

- QA based architectures are a better choice to minimize energy for low-SNR applications. Meanwhile, QS based architectures are a better choice to operate with higher SNR requirements, e.g., Fig. 4.6 shows that QA-BB is more energy-efficient than QS-BM for SNR$<$10 dB.

- The dot product size N can be traded off with SNR for a given array in QA-BB architecture. Therefore, a dot product can be partitioned across multiple banks in order to boost SNR. Additionally, QA-BB architecture allows a variety of input precision and weight precision on the same hardware. This makes QA-BB a suitable architecture to exploit energy-vs-accuracy trade-offs via compiler-driven methods such as those described in [54].

- ADC quantization step size should decrease with the dot product size N for QS-based architectures. A rule of thumb in choosing the ADC step size is to set $\Delta_{\text{adc}} = \sqrt{G/N}$, where G is a constant that depends on the architecture, data statistics, and the SNR, e.g., it can be obtained from (4.14) and (4.7).

4.4 Conclusions

In this chapter, we analyzed the precision and accuracy limits of in-memory architectures. We studied three in-memory architectures and found that: (a) the SNR of in-memory dot products is often limited by circuit non-idealities, (b) large vector lengths lead to ADC challenges in QS architectures, and (c) clipping in QA-based operations results can limit dot product lengths.

The ADC challenge in QS due to the decrease in V_r with N can be partially mitigated by exploiting sparsity in inputs $\mathbf{x}$ during charge sharing, as in [35, 56]. The effects of circuit non-idealities in these in-memory architectures can be mitigated through circuit and algorithmic techniques. For example, the effects of charge injection noise in QS were mitigated via calibration in [31] and through the use of additional switches in [104]. Redundancy and error compensation techniques could also be explored to address accuracy limitations. We study such techniques in Chapter 5 and Chapter 6 where on-chip learning and data encoding methods are used to address circuit non-idealities, respectively.

Derivations of Equations (4.1), (4.2), (4.7), (4.8), (4.9), (4.12), and (4.14)

Derivation of (4.1):
In the CM architecture, clipping occurs when weights are read from memory. We can therefore write:

$$\eta_c = {\lambda_{\mathbf{w}}}^T \mathbf{x}$$

where $\lambda_{\mathbf{w}}$ is a vector of clipping noise terms, each element of which represents an additive noise corrupting each weight due to clipping. The clipping noise terms can be assumed to be independent from each other and from the inputs.

Furthermore, as discussed next, the clipping noise is assumed to have zero mean, so that:

$$\sigma_{\eta_c}^2 = N\mathbb{E}\left[x^2\right]\mathrm{Var}(\lambda)$$

where the clipping noise λ is modeled as a mixture of three independent random variables as follows: We assume the λ is zero mean uniformly distributed between $[-(w_{\max} - w_{\text{clip}}), w_{\max} - w_{\text{clip}}]$ given $|w| > w_{\text{clip}}$. Furthermore, $p = \Pr(|w| \geq w_{\text{clip}}) \leq \frac{\sigma_w^2}{w_{\text{clip}}^2}$ by virtue of Chebyshev's inequality, $w_{\text{clip}} = k_{\text{clip}}\Delta_w$ and $w_{\max} = 1$ by definition. Under these conditions, it can be shown that

$$\mathrm{Var}(\lambda) \approx \frac{1}{12}\sigma_w^2 k_{\text{clip}}^{-2} 2^{2B_w}\left(1 - 2k_{\text{clip}}2^{-B_w}\right)_+^2$$

so that (4.1) follows.

Derivation of (4.2):

In CM, circuit non-idealities manifest during the discharge of weight bits. Thus,

$$\sigma_{\eta_e}^2 = N\mathbb{E}\left[x^2\right]\mathrm{Var}(\delta)$$

where δ is the noise term due to circuit non-idealities per weight discharge. Due to the nature of one's complement representation used in CM, it can be shown that:

$$\mathrm{Var}(\delta) = \sum_{i=1}^{B_w - 1} 4^{-i}\mathbb{E}[w_i^2]\sigma_{\text{D}}^2$$

and assuming the weight bits are equally likely to be 0 or 1, (4.2) follows.

Derivation of (4.7):

In CM, BL discharge is given by:

$$\Delta V_{\text{BL}} = 2^{B_W} V_{\text{BL,LSB}} w_i$$

and therefore after multiplications and aggregations via charge sharing voltage at he input of the ADC is given by:

$$\Delta V_o = \frac{2^{B_W} V_{\text{BL,LSB}}}{N}\sum_i^N w_i x_i$$

Therefore,

$$V_{\text{r}} = 4\frac{2^{B_W} V_{\text{BL,LSB}}}{N}\sigma_y$$

Since $\sigma_y = \sqrt{N}\mathbb{E}[x^2]\sigma_w^2$, we get (4.7).

Derivation of (4.8):

From the nature of the arithmetic in the QA-BB architecture, we have:

$$\sigma_{\eta_c}^2 = \sum_{i=1}^{B_w}\sum_{j=1}^{B_x} 4^{1-i-j}\mathbb{E}\left[\lambda_{i,j}^2\right]$$

where $\lambda_{i,j}$ is the clipping noise term for every bit-wise dot product. Note that the nature of two's complement arithmetic makes the overall clipping noise zero-mean in spite of the individual clipping noise terms being non zero-mean. Assuming the clipping noise terms are independent and identically distributed, we obtain:

$$\sigma_{\eta_c}^2 = \frac{4}{9}\mathbb{E}\left[\lambda^2\right]\left(1-4^{-B_w}\right)\left(1-4^{-B_x}\right)$$

where the clipping noise term is $\lambda = (y_{\text{BL}} - y_{\text{clip}})\mathbb{1}_{\{y_{\text{BL}} > y_{\text{clip}}\}}$ where y_{BL} is the discharge on a bit-line per bit-wise dot product. Assuming weight and input bits to be independent and equally likely to be 0 or 1, we obtain that y_{BL} follows a binomial distribution $Bi(\frac{1}{4})$ so that:

$$\mathbb{E}\left[\lambda^2\right] = \sum_{k=k_{\text{clip}}}^{N}(k-k_{\text{clip}})^2\binom{N}{k}\left(\frac{1}{4}\right)^k\left(\frac{3}{4}\right)^{N-k}$$

and thus (4.8) follows.

Derivation of (4.9):

In the QA-BB architecture circuit non-idealities are manifested whenever a bit-cell is discharged, i.e., whenever an input and corresponding weight bit are equal to 1. From the nature of the arithmetic in the QA-BB architecture, we obtain:

$$\sigma_{\eta_e}^2 = \sum_{i=1}^{B_w}\sum_{j=1}^{B_x}\sum_{k=1}^{N} 4^{1-i-j}\text{Var}(\delta_{i,j,k})$$

where $\delta_{i,j,k}$ is the noise term due to circuit non-idealities which occurs when accessing the bit-cell at location (i,k) during the j^{th} cycle. Assuming the noise terms are independent and identically distributed, we obtain:

$$\sigma_{\eta_e}^2 = \frac{4}{9}N\left(1-4^{-B_w}\right)\left(1-4^{-B_x}\right)\text{Var}(\delta)$$

where δ is the circuit noise per bit-cell discharge whose variance equals:

$$\mathrm{Var}(\delta) = \frac{1}{4}\sigma_{\mathrm{D}}^2$$

where the $\frac{1}{4}$ term is due to the necessity of both input and weight bits to equal 1. Thus, (4.9) follows.

Derivation of (4.12):

From noise model of QS compute model(3.25) and BB decomposition (3.36) we have we have:

$$\sigma_{\eta_e}^2 = \sum_{i=1}^{B_w} 4^{1-i-j}\mathbb{E}\left[\left(\frac{Nv_e}{V_{\mathrm{dd}}}\right)^2\right]$$

$$\approx \sum_{i=1}^{B_w} 4^{1-i}\mathbb{E}\left[\left(\frac{\sum_{j=1}^{N}\left(Nx_jw_{i,j}c_jV_{\mathrm{dd}} + NC_{\mathrm{o}}v_j + NC_{\mathrm{o}}V_{\mathrm{dd}}x_jw_{i,j} + Nv_\theta C_{\mathrm{o}}\right)}{\sum_{j=1}^{N}(C_{\mathrm{o}}+c_j)V_{\mathrm{dd}}} - \sum_{j=1}^{N} x_jw_{i,j}\right)^2\right]$$

Neglecting the noise term in the denominator as it is averaged with N, and assuming uncorrelated random variables we get the following:

$$\sigma_{\eta_e}^2 \approx \sum_{i=1}^{B_w} 4^{1-i}\mathbb{E}\left[\left(\frac{\sum_{j=1}^{N}(x_jw_{i,j}c_jV_{\mathrm{dd}} + C_{\mathrm{o}}v_j + v_\theta C_{\mathrm{o}})}{C_{\mathrm{o}}V_{\mathrm{dd}}}\right)^2\right]$$
$$= \sum_{i=1}^{B_w} 4^{1-i}\frac{\sum_{j=1}^{N}(V_{\mathrm{dd}}^2\mathbb{E}[x_j^2]\mathbb{E}[w_{i,j}^2]\mathbb{E}[c_j^2] + C_{\mathrm{o}}^2\mathbb{E}[v_j^2] + \mathbb{E}[v_\theta^2]C_{\mathrm{o}}^2)}{C_{\mathrm{o}}^2V_{\mathrm{dd}}^2}$$

Substituting capacitor mismatch variance $\sigma_{C_{\mathrm{o}}}^2$, and thermal noise variance σ_θ^2 from (3.26) and (3.28), respectively, we get the following:

$$\sigma_{\eta_e}^2 = \sum_{i=1}^{B_w} 4^{1-i}\frac{NV_{\mathrm{dd}}^2\mathbb{E}[x^2]0.5\sigma_{C_{\mathrm{o}}}^2 + C_{\mathrm{o}}^2\mathbb{E}[v_j^2] + \sigma_\theta^2C_{\mathrm{o}}^2}{C_{\mathrm{o}}^2V_{\mathrm{dd}}^2}$$

and thus follows (4.12):

$$\sigma_{\eta_{\mathrm{e}}}^2 = \frac{2}{3}(1-4^{-B_w})N\left(\frac{\mathbb{E}\left[x^2\right]\sigma_{C_{\mathrm{o}}}^2}{C_{\mathrm{o}}^2} + \frac{2\sigma_\theta^2}{V_{\mathrm{dd}}^2}\right) + \sigma_{\mathrm{inj}}^2$$

where the charge injection noise term $\sigma_{\text{Inj}} = \sqrt{N\mathbb{E}\left[x^2\right]}\sigma_w WLC_{\text{OX}}/C_{\text{o}}$.

Derivation of (4.14):

In QS-BM, we estimate binary-weighted dot-product in each column using charge sharing, as follows:

$$V_i = \frac{V_{\text{dd}}}{N}\sum_{j}^{N} x_j w_{i,j}$$

Since in V_i is the ADC input in QS-BM we need its standard deviations to estimate V_{r}. Since $w_{i,j}$ is binary-valued,

$$\mathbb{E}[V] = V_{\text{dd}}E[x] = 0.5V_{\text{dd}}\mu_x$$

$$\mathbb{E}[(V - 0.5V_{\text{dd}}\mu_x)^2] = \frac{V_{\text{dd}}^2}{4N}(2E[x_j^2] - \mu_x^2)$$

Since $V_{\text{r}} = 8\sqrt{\mathbb{E}[V^2] - \mathbb{E}[V]^2}$, we get (4.14).

CHAPTER 5

VARIATION-TOLERANT IN-MEMORY ARCHITECTURES VIA ON-CHIP LEARNING

DIMA IC realizations [55, 96] by implementing low-swing analog computations on the BLs have demonstrated $> 50\times$ EDP gains over their digital counterparts. However, their analog nature combined with stringent area constraints makes computations on DIMA susceptible to PVT variation and other circuit non-idealities. Indeed, the analysis in Chapter 4 characterizes the impact of DIMA's non-ideal analog behavior on the SNR of the computations, where we observe that the SNR of DIMA trades off with its energy consumption and latency. For example, in compute memory as per (3.42), reducing the BL voltage discharge ΔV_{BL} reduces the energy consumption and the SNR of the dot-products simultaneously. Therefore, if DIMA's EDP gains are to be enhanced even further, one needs to develop algorithmic techniques to overcome the impact of reduced compute SNR.

This chapter[1] studies the use of an on-chip training [49] in order to overcome the impact of circuit non-idealities and hence enhance DIMA's EDP gains. Since ML algorithms employ data-driven training methods to learn sufficient statistics for accurate inference, it is possible to harness the power of such methods to realize an on-chip learning set-up whereby the training method adapts to both data statistics and the statistics of non-ideal circuit behavior such as those due to process variations. The use of on-chip learning is demonstrated using a 65 nm CMOS IC prototype that realizes a robust deep in-memory support vector machine (SVM) classifier IC using a SGD-based on-chip trainer. SGD-based training [105] was chosen since it is commonly employed to train many ML systems, including deep neural networks (DNNs), leading to the possibility of other applications of this work.

[1]Adapted from: Sujan K. Gonugondla, Mingu Kang, and Naresh R. Shanbhag, "A variation-tolerant in-memory machine learning classifier via on-chip training" *IEEE Journal of Solid-State Circuits (JSSC)*, 2018, © IEEE. [49]

5.1 Background

This section provides background the SVM algorithm, and its training via the SGD algorithm.

5.1.1 Stochastic Gradient Descent (SGD)

Stochastic gradient descent (SGD) has emerged as one of the most effective training algorithms for machine learning [105, 106]. The SGD algorithm has been used to train a variety of ML algorithms including the support vector machine (SVM) [107]. The SVM is a maximum margin binary classifier which predicts the class label $y_k \in \{\pm 1\}$ as follows:

$$\mathbf{W}^T\mathbf{X}_k + b \underset{\hat{y}_k=-1}{\overset{\hat{y}_k=+1}{\gtrless}} 0 \tag{5.1}$$

where $\hat{y}_k$ is the predicted class label, $\mathbf{X}_k = [X_{0,k}, ..., X_{D-1,k}]^T$ is the D-dimensional input vector, and weights $\mathbf{W} = [W_0, ..., W_{D-1}]^T$ and a scalar bias b are referred to as the parameters of the SVM algorithm. The SVM cost function per sample $Q(\mathbf{W}, \mathbf{X}_k)$ is given by [105]:

$$Q(\mathbf{W}, \mathbf{X}_k) = \frac{\lambda}{2}(||\mathbf{W}||^2 + b^2) + [1 - y_k(\mathbf{W}^T\mathbf{X}_k + b)]_+ \tag{5.2}$$

where λ is a regularization factor, y_k is the true label for the data sample $\mathbf{X}_k$, and $[x]_+ = \max\{0, x\}$. The SGD algorithm minimizes the cost function $Q(\mathbf{W}, \mathbf{X}_k)$ averaged over the training set iteratively by updating the weight vector as follows:

$$\begin{aligned}\mathbf{W}_{m+1} &= (1 - \lambda\gamma)\mathbf{W}_m \\ &+\gamma\frac{1}{N}\sum_{n=0}^{N-1}\begin{cases}0 & \text{if } y_n^{(m)}(\mathbf{W}_m^T\mathbf{X}_n^{(m)} + b_m) > 1 \\ y_n^{(m)}\mathbf{X}_n^{(m)} & \text{otherwise}\end{cases}\end{aligned} \tag{5.3}$$

$$\begin{aligned}b_{m+1} &= (1 - \lambda\gamma)b_m \\ &+\gamma\frac{1}{N}\sum_{n=0}^{N-1}\begin{cases}0 & \text{if } y_n^{(m)}(\mathbf{W}_m^T\mathbf{X}_n^{(m)} + b_m) > 1 \\ y_n^{(m)} & \text{otherwise}\end{cases}\end{aligned} \tag{5.4}$$

where N is the batch size, m is the batch index, $\mathbf{W}_m$ is the weight vector in the m-th batch, $\mathbf{X}_n^{(m)}$ is the n-th sample of the m-th batch, and γ is the learning rate.

The primary inference computation in the SVM algorithm is the dot product $\mathbf{W}^T\mathbf{X}$. For SVM implementation on DIMA, the weights $\mathbf{W}$ are stored in the BCA and accessed via functional read, while the BLPs execute element-wise multiplication of $\mathbf{W}$ and $\mathbf{X}$, and the CBLP aggregates the BLP outputs to obtain the final dot product.

5.2 A Systems Rationale for On-Chip Training

This section provides a systems rationale for using the SGD algorithm on-chip to compensate for PVT variations in DIMA. These variations are caused by [28]: (1) spatial transistor threshold voltage (V_t) variations caused by random dopant fluctuations [29]; (2) BL voltage dependence of the discharge path (access and pull-down transistors in the bit-cell) current; and (3) the finite transition (rise and fall) times of the PWM WL pulses. As a result, the functional read step of DIMA, which accesses weights W_j stored in the j-th column, generates a BL discharge of $\Delta V_{\mathrm{BL},j} = \alpha_j W_j$ on the j-th BL. Therefore, even though the BCA stores $\mathbf{W} = [W_0, W_1, ..., W_{D-1}]$, DIMA implements the dot product in (5.1) with weights $\mathbf{W'} = [\beta_0 W_0, \beta_1 W_1, ..., \beta_{D-1} W_{D-1}]$, where $\beta_j = \frac{\alpha_j}{\alpha}$ (with $\alpha_j = \alpha(1 + \frac{\Delta\alpha_j}{\alpha})$) is the per dimension proportionality factor. Measured results show that this simple model captures the effects of spatial PVT variations to a first order.

Thus, the SGD algorithm minimizes the modified cost function $Q'(\mathbf{W}, \mathbf{X}_k)$ given by:

$$\begin{aligned} Q'(\mathbf{W}, \mathbf{X}_k) &= \frac{\lambda}{2}(||\mathbf{W'}||^2 + b^2) + [1 - y(\mathbf{W'}^T\mathbf{X}_k + b)]_+ \\ &= \frac{\lambda}{2}(\sum_{j=0}^{D-1} \beta_j^2 W_j^2 + b^2) + [1 - y(\sum_{j=0}^{D-1} \beta_j W_j X_{j,k} + b)]_+ \end{aligned} \tag{5.5}$$

The modified cost function $Q'(\mathbf{W}, \mathbf{X})$ can be shown to be convex in $\mathbf{W}$, which implies that the SGD algorithm converges to the global minimum in the presence of PVT variations. However, as the proportionality factor β_j is unknown in practice and is die-specific, we employ a per dimension learning

rate of $\gamma_j = \gamma/\beta_j$ to obtain following SGD update:

$$\begin{aligned}\mathbf{W}_{m+1} &= (1-\lambda\gamma)\mathbf{W}_m \\ &+\gamma\frac{1}{N}\sum_{n=0}^{N-1}\begin{cases}0 & \text{if } y_n^{(m)}(\mathbf{W}_m'^T\mathbf{X}_n^{(m)}+b_m)>1\\ y_n^{(m)}\mathbf{X}_n^{(m)} & \text{otherwise}\end{cases}\end{aligned} \tag{5.6}$$

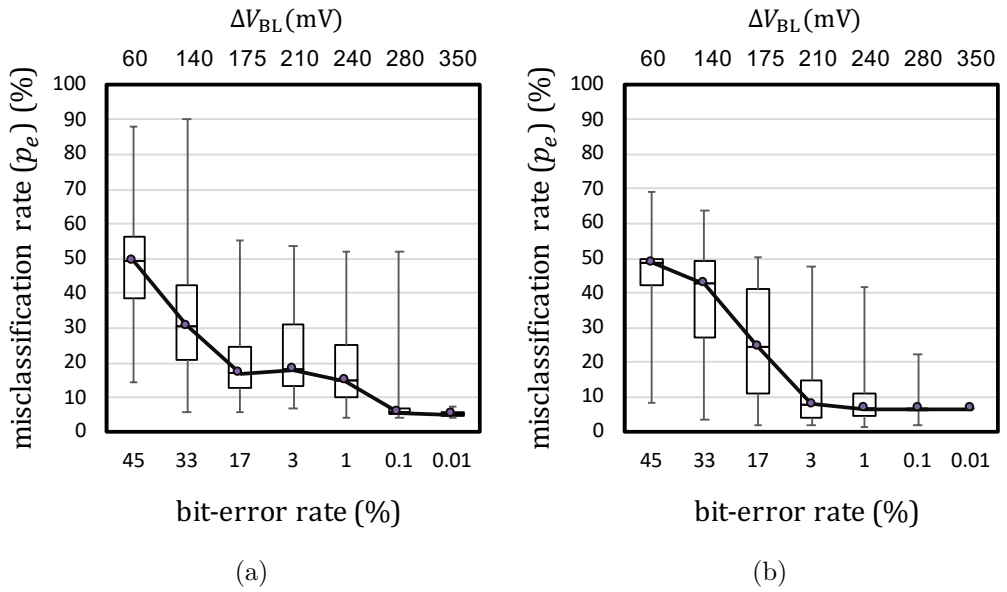
which is identical to (5.3) except that it relies upon computation of the dot product in the presence of spatial variations. In conventional digital architectures, on-chip training is ineffective in compensating for PVT variations under reduced ΔV_{BL}. This is because PVT variations with reduced ΔV_{BL} lead to increased sense amplifier bit errors including the most significant bit (MSB) errors. These large magnitude errors in turn lead to a large increase in the mean and variance of the misclassification rate over different instances of the architecture. Figure 5.1 shows that the increase in the misclassification rate cannot be compensated for via on-chip learning. DIMA avoids such errors by reading a weighted function of the bits of W instead of the bits directly.

5.3 Implementation

The architecture of the prototype IC in Fig. 5.2 has four major blocks: (a) the in-memory CORE (IM-CORE) to execute the inference computations, (b) the standard read/write SRAM interface, (c) the digital trainer, and (d) a digital controller (CTRL) to sequence the operations. The prototype IC has three modes of operation: (1) a standard SRAM mode, (2) an in-memory inference mode, and (3) a training mode.

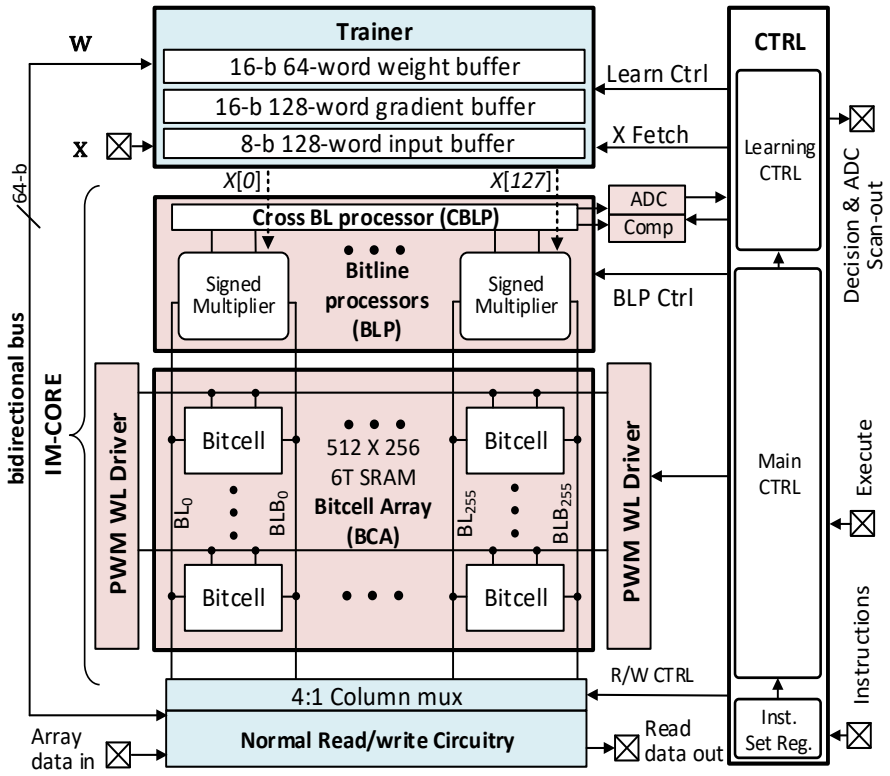
5.3.1 The IM-CORE Block

The in-memory inference mode is implemented by the IM-CORE block. The IM-CORE comprises a conventional 512×256 6T SRAM BCA, and in-memory computation circuitry, which includes: (a) pulse width modulated WL drivers to realize functional read, (b) BLPs implementing signed multiplication, (c) CBLP implementing summation, (d) an ADC bank, and (e) a comparator bank to generate final decisions. The 16-b SVM weights $\mathbf{W}$ are

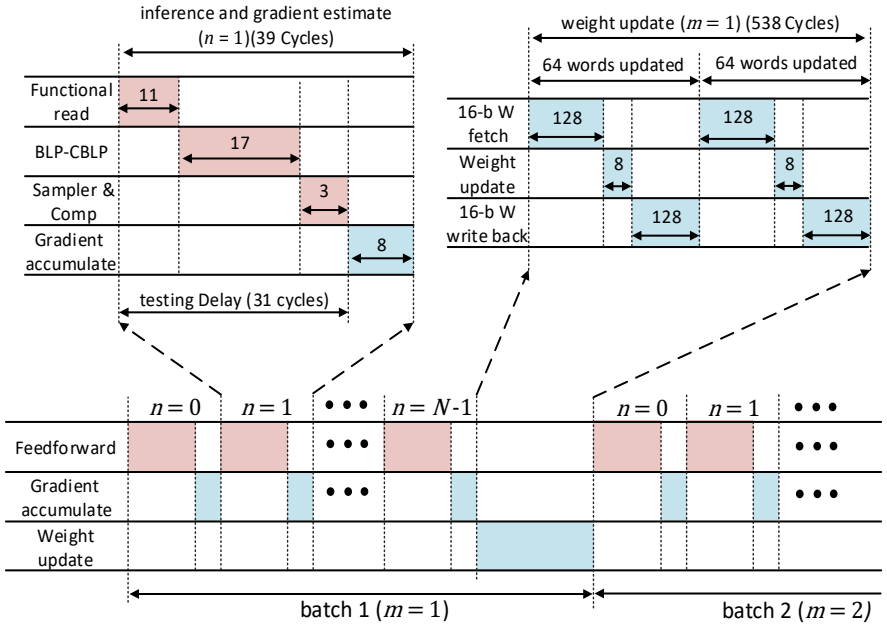


(a) (b)

Figure 5.1: Misclassification rate of a digital architecture when subject to bit errors during readout: (a) before retraining, and (b) after retraining. The bit error rate (BER) was obtained via measurements from the prototype IC under reduced BL swing (ΔV_{BL}) in the SRAM mode. The misclassification rate was obtained via simulations of 1000 independent instances of an SRAM bank, with each instance processing 858 data samples, in presence of randomly assigned bit-flip locations at a rate based on the BER.



(a)



(b)

Figure 5.2: Proposed system: (a) chip architecture showing IM-CORE, trainer and CTRL, and (b) the timing diagram.

stored in the BCA. However, only 8-b MSBs of the weights **W** are used during inference (see Fig. 5.3). Functional read simultaneously generates ΔV_{BL} discharge voltages on the BLs in one read cycle. The 8-b input samples **X** are streamed into the input buffer, and are transferred to the BLPs via a 256-b bus for element-wise multiplication of **W** and **X**, which are then summed in the CBLP to obtain the dot product in (5.1).

Signed functional read

Functional read performs digital-to-analog conversion of the weights stored in the SRAM array such that the discharge on the BL represents the analog values of the weights being read.

The 8-b SVM weights $W \equiv \{w_7, .., w_1, w_0\}$ ($w_i \in \{0, 1\}$ is the i-th binary bit of W) are stored in a column-major format split across two adjacent columns (MSB and LSB columns) with 4-b per column in the BCA as shown in Fig. 5.3. The application of WL access pulses with binary weighted pulse widths (PWM) followed by LSB-MSB merge results in discharge voltages on BL and BLB as follows:

$$\Delta V_{\mathrm{BL}} = \frac{V_{\mathrm{pre}} T_0}{R_{\mathrm{BL}} C_{\mathrm{BL}}} \left[\frac{1}{16} \sum_{i=0}^{3} 2^i \overline{w}_i + \sum_{i=4}^{7} 2^i \overline{w}_i \right] \tag{5.7}$$

$$\Delta V_{\mathrm{BLB}} = \frac{V_{\mathrm{pre}} T_0}{R_{\mathrm{BL}} C_{\mathrm{BL}}} \left[\frac{1}{16} \sum_{i=0}^{3} 2^i w_i + \sum_{i=4}^{7} 2^i w_i \right] \tag{5.8}$$

where V_{pre} is the BL precharge voltage, T_0 is the least significant bit (LSB) pulse width, C_{BL} is the BL capacitance, and R_{BL} is the BL discharge path resistance of a bit-cell. We employ two's complement representation for W to account for its sign. The magnitude $|W|$ is obtained by choosing ΔV_{BLB} (ΔV_{BL}) if W is negative (positive) since the two discharges are complementary. The sign of W is obtained by comparing ΔV_{BLB} and ΔV_{BL} as shown in Fig. 5.3. Thus, the discharge on the output node of functional read $\Delta V_{\mathrm{BL_{MUX}}} \propto |W|$. The detected sign and the magnitude are then passed on to the signed multipliers in the BLP.

Equation (5.8) assumes that R_{BL} is invariant with respect to the time-varying ΔV_{BL} and spatially across the BCA. In practice, the amplitude of word-line voltages V_{WL} is kept sufficiently low (0.45 V to 0.65 V) to alleviate the impact of ΔV_{BL} on R_{BL}. Doing so results in the integral non-linearity

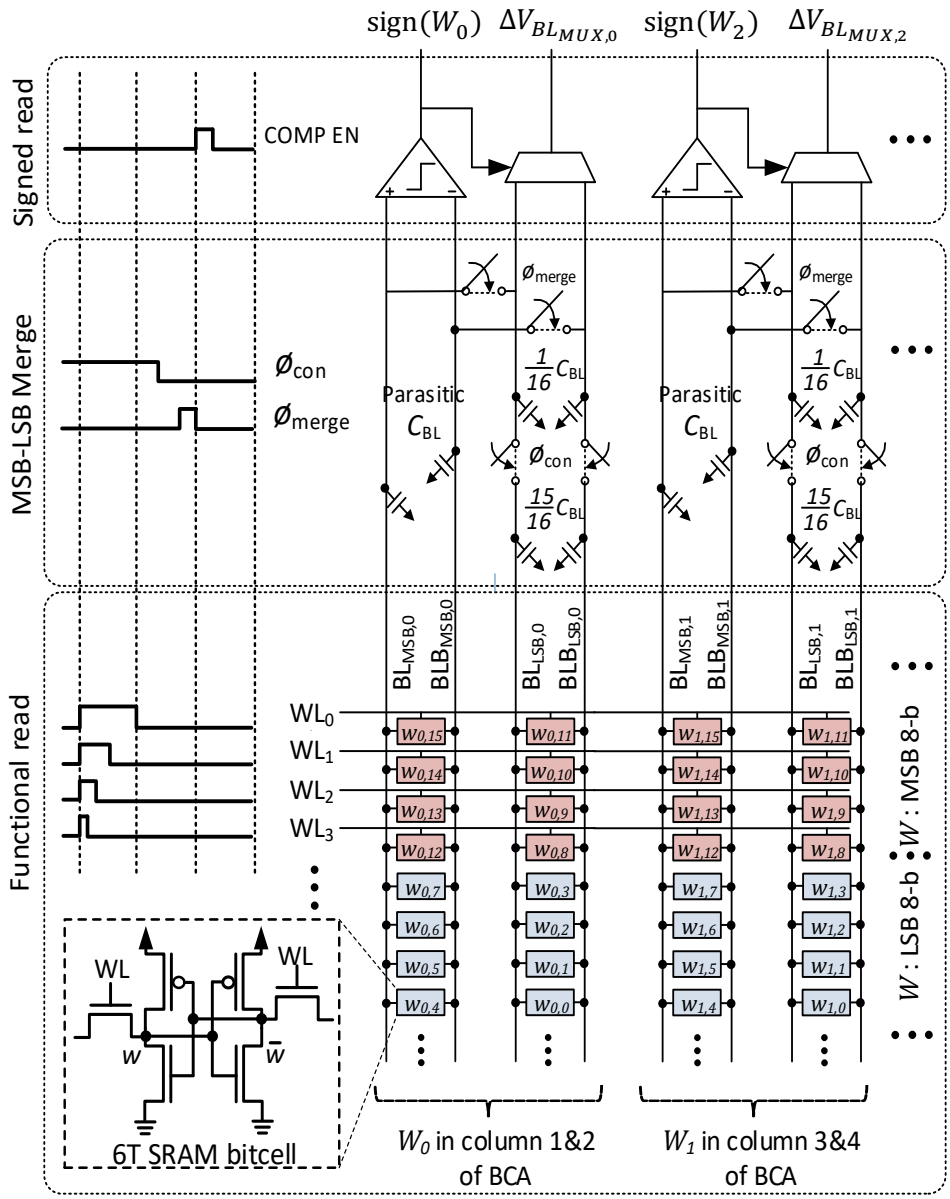
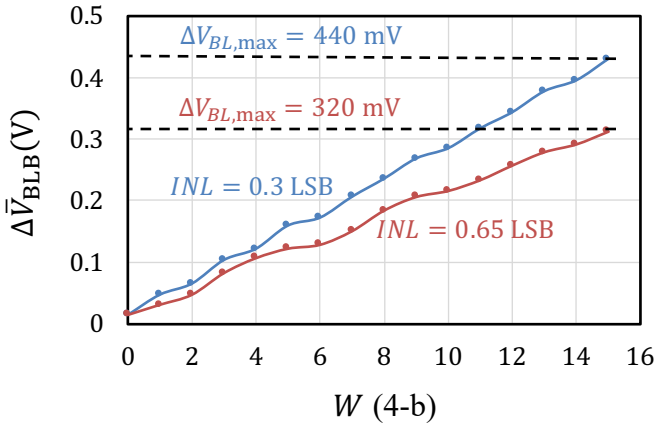
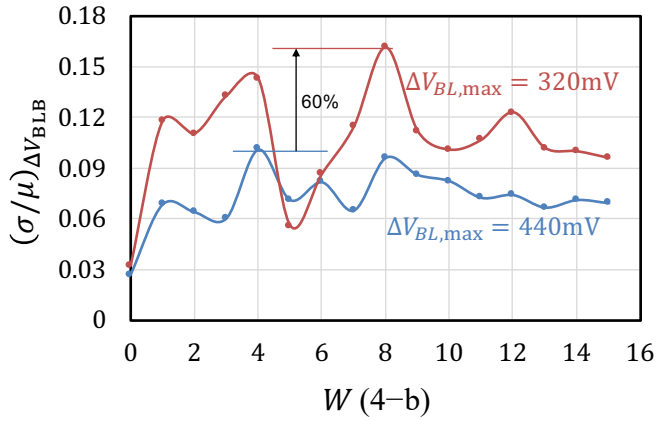


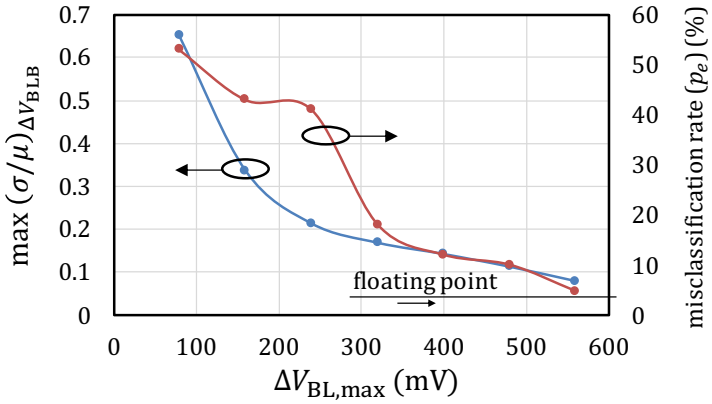
Figure 5.3: Implementation of signed functional read. The 8 MSBs of W are stored across two adjacent bit-cell columns (red) and employed for inference, while 8 LSBs of W (blue) are updated during training.



(a)



(b)



(c)

Figure 5.4: Impact of spatial variations on functional read obtained by measurements across 30 randomly chosen 4-row groups: (a) average $\Delta\overline{V}_{\mathrm{BLB}}$, (b) normalized variance $(\sigma/\mu)_{\Delta V_{\mathrm{BLB}}}$, and (c) the impact of spatial variations on $(\sigma/\mu)_{\Delta V_{\mathrm{BLB}}}$ and misclassification rate p_e with respect to $\Delta V_{\mathrm{BL,max}}$.

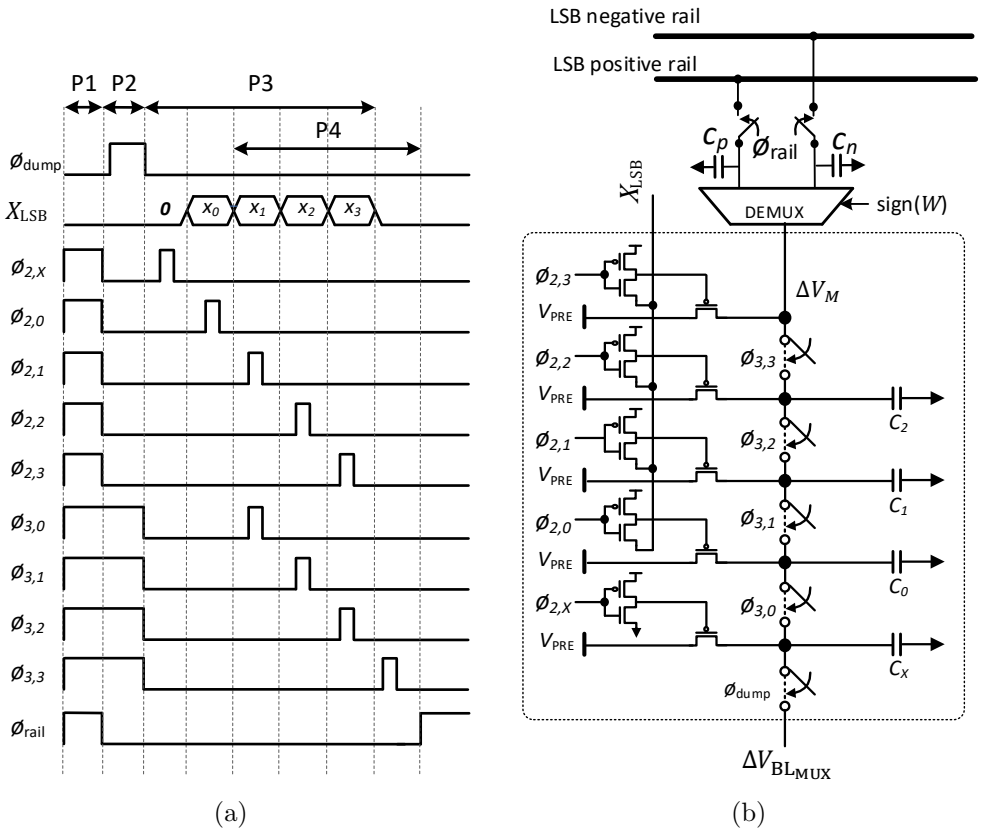
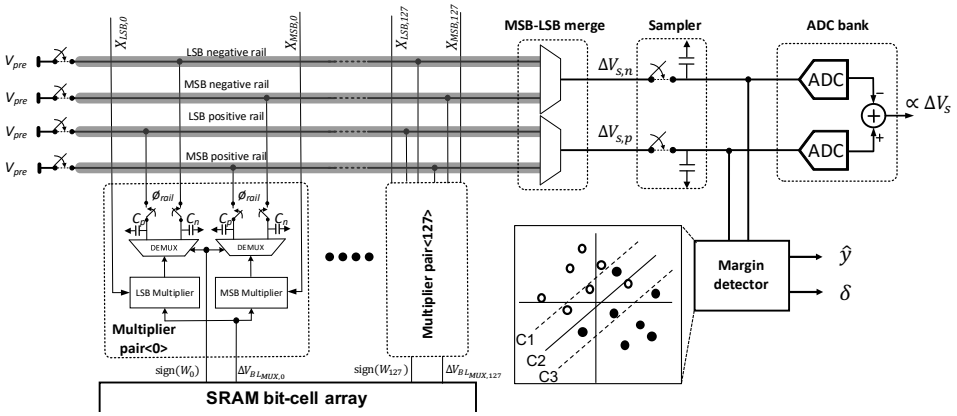
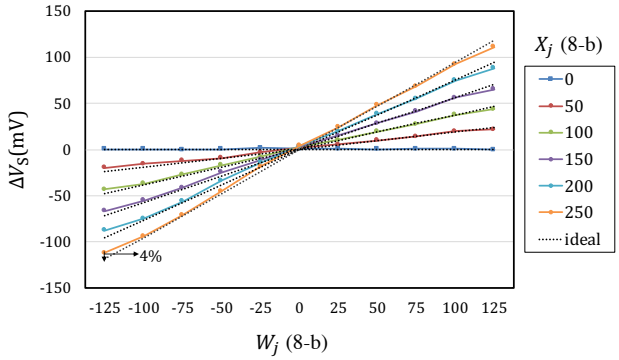


Figure 5.5: The signed 4-b×8-b LSB multiplier: (a) timing diagram, and (b) circuit schematic.



(a)



(b)

Figure 5.6: Aggregation in the CBLP: (a) circuit schematic, and (b) measured output using identical values of W_j and X_j across all the BCA columns, at $\Delta V_{\mathrm{BL,max}} = 320\,\mathrm{mV}$.

(INL) of ΔV_{BLB} to be less than 0.65 LSB (see Fig. 5.4(a)).

Spatial variations in the access transistor threshold voltages lead to variations in R_{BL}, and hence to variations in ΔV_{BL} across the BCA columns. Figure 5.4(b) shows that the variance of ΔV_{BLB} at $\Delta V_{\mathrm{BL,max}} = 320\,\mathrm{mV}$ is 60% higher than at $\Delta V_{\mathrm{BL,max}} = 440\,\mathrm{mV}$. Figure 5.4(c) shows that the misclassification rate $p_e = 4\%$ at $\Delta V_{\mathrm{BL,max}} = 560\,\mathrm{mV}$ is within 1% of the floating point accuracy. However, p_e increases to 16% when $\Delta V_{\mathrm{BL,max}}$ is reduced to 320 mV, illustrating the impact of spatial variations in ΔV_{BL} on the accuracy of inference. The impact of spatial variations on ΔV_{BL} can be reduced by increasing the maximum BL discharge voltage $\Delta V_{\mathrm{BL,max}}$ (ΔV_{BLB} when the 4-b $W = 15$) by tuning the WL voltage V_{WL} and T_0.

This sensitivity to chip-specific spatial process variations indicates the need to explore on-chip compensation methods. Additionally, increasing $\Delta V_{\mathrm{BL,max}}$ to reduce the impact of spatial variations incurs an energy cost thereby leading to an interesting trade-off between computational accuracy and energy in DIMA that can be exploited at the architectural level.

BLP: Signed multiplication

The BLP block needs to implement a signed 8-b×8-b element-wise product ($W_i X_i$) between **W** and **X**. The BLP block realizes this using a MSB and LSB multiplier pair with each multiplying the functional read output $\Delta V_{\mathrm{BL_{MUX}}} \propto$ 8-b $|W|$ with four MSBs (X_{MSB}) and four LSBs (X_{LSB}) of an 8-b input X. The proposed charge-domain signed multiplier in Fig. 5.5 is based on the unsigned version in [28].

The LSB multiplier receives digital inputs $X_{\mathrm{LSB}} \equiv \{x_3, .., x_0\}$ ($x_i \in \{0, 1\}$ is i-th bit of X), the analog $\Delta V_{\mathrm{BL_{MUX}}}$, and a digital sign($W$) from functional read. Multiplication occurs in four phases P1-P4 with P3 and P4 overlapping in time (see Fig. 5.5(a)). The multiplier (Fig. 5.5(b)) employs six equally sized 25 fF capacitors (C_0, C_1, C_2, C_x, C_p, C_n) which are initialized to V_{pre} and either C_p or C_n is chosen as the output capacitor based on sign(W) (P1). Next, the five capacitors nodes are charge shared with node $\Delta V_{\mathrm{BL_{MUX}}}$ using the $\phi_{3,i}$ and ϕ_{dump} switches (P2). This is followed by conditionally charging capacitors C_i to V_{pre} using the $\phi_{2,i}$ switches (P3). Capacitors C_x, C_0, C_1, and C_2, are sequentially charge shared using the $\phi_{3,i}$ switches (P4), resulting in

$$\Delta V_{\mathrm{M}} = \frac{\Delta V_{\mathrm{BL_{MUX}}}}{16} \sum_{i=0}^{3} 2^i x_i \propto |X_{\mathrm{LSB}}||W| \tag{5.9}$$

where ΔV_{M} is the discharge on the output capacitor C_p or C_n connected to the positive and negative output rails (see Fig. 5.5(b)). The MSB multiplier operates identically.

Aggregation and final decision

The CBLP aggregates the outputs of the MSB and LSB BLP multipliers from across the BCA in order to generate the dot product $\mathbf{W}^T\mathbf{X}$. This aggregation is accomplished (Fig. 5.6(a)) by merging the positive (negative) rails from both the MSB and LSB multipliers across the BCA with a 16:1 charge sharing ratio (see Fig. 5.6(a)). This merging step results in the voltages $\Delta V_{\mathrm{S,p}}$ and $\Delta V_{\mathrm{S,n}}$ representing the magnitude of the sum of the positive and negative terms, respectively, in the final dot product. Therefore, the dot product is computed as:

$$\Delta V_{\mathrm{S}} = \Delta V_{\mathrm{S,p}} - \Delta V_{\mathrm{S,n}} \propto \mathbf{W}^T\mathbf{X} \tag{5.10}$$

The voltages $V_{\mathrm{S,p}}$ and $V_{\mathrm{S,n}}$ are sampled and processed by a bank of three comparators (C1-C3) where C2 generates the predicted label ($\hat{y}$) while C1 and C3 realize the SVM *margin detector* (see Section 5.3.2). The sampled rail voltages $V_{\mathrm{S,p}}$ and $V_{\mathrm{S,n}}$ are also converted to digital via a 6-b ADC pair for testing purposes. Measured (Fig. 5.6(b)) values of ΔV_{S} are found to lie within $< 4\%$ of the dynamic range when $\Delta V_{\mathrm{BL,max}} = 320\,\mathrm{mV}$.

5.3.2 Trainer

As the energy and latency cost of SRAM writes are very high, the trainer, which is implemented digitally, writes the updated weights once per batch into the BCA. The trainer (see Fig. 5.2) implements a reformulated version of the batch update (5.6) shown below:

$$\mathbf{W}_{m+1} = (1 - \gamma\lambda)\mathbf{W}_m + \frac{\gamma}{N}\Delta_{W,N}^{(m)} \tag{5.11}$$

where $\Delta_{W,N}^{(m)}$ is the *batch gradient estimate* generated by accumulating the *sample gradient estimate* $\Delta_{W,n}^{(m)}$:

$$\Delta_{W,n+1}^{(m)} = \Delta_{W,n}^{(m)} + \begin{cases} y_n^{(m)}\mathbf{X}_n^{(m)} & \text{if } \delta_n^{(m)} \leq 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.12}$$

where $y_n^{(m)}$ is the true label corresponding to the data sample $\mathbf{X}_n^{(m)}$, and $\delta_n^{(m)}$ is computed in the margin detector (see Fig. 5.6(a)) as follows:

$$\delta_n^{(m)} = y_n^{(m)}[\text{sign}(z_n^{(m)} - 1) + \text{sign}(z_n^{(m)} + 1)] \tag{5.13}$$

and $z_n^{(m)} = \mathbf{W}_m'^T \mathbf{X}_n^{(m)} + b$ is the output of the CBLP block. The margin detector also generates the SVM decision $\hat{y}_n^{(m)} = \text{sign}(z_n^{(m)})$.

The trainer consists of an input buffer to store the streamed input $\mathbf{X}$ and a gradient buffer to store $\Delta_{W,n}$. The trainer implements (5.11) and (5.12) by reusing 64 16-b adders to conserve area. The weight update is performed in two cycles, where 64 words of $\mathbf{W}$ are updated per cycle (see Fig. 5.2(b)). Multiplications with γ/N and $\gamma\lambda$ in (5.11) are implemented using barrel shifters, thereby restricting them to powers of 2 in the range $[1, 2^{-15}]$. The ability to choose learning rates in powers-of-2 provides a wider tuning range over learning rates chosen on a linear scale. Wider tuning range in fact allows operating with very small learning rates, thereby enabling fine-tuning of the weights to achieve a lower misclassification rate.

Trainer precision assignment

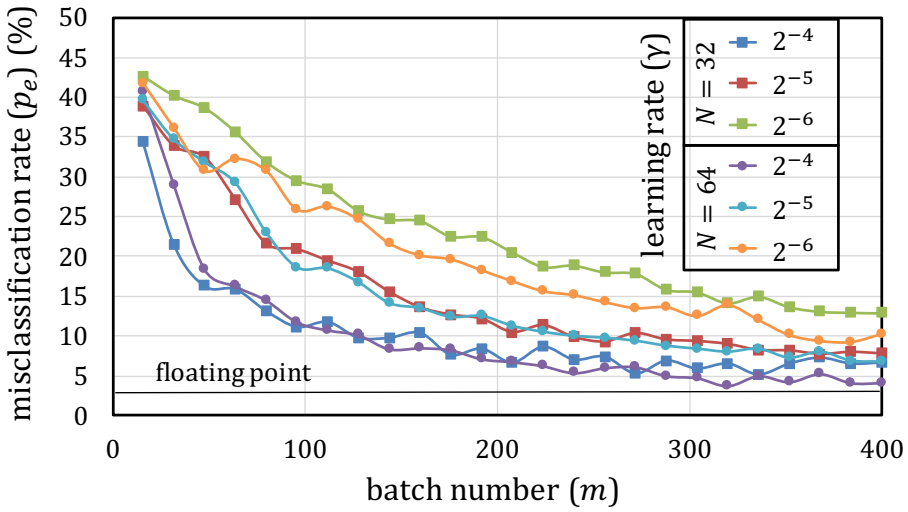
The precision of the trainer needs to be chosen carefully in order to minimize the cost of training without compromising the convergence behavior of the SGD algorithm. The minimum bit precision for W and $\Delta_{W,n}$ needs to be set. Additionally, the minimum precision for W during inference (5.1) and training (5.11), denoted by B_W and B_{WUD}, respectively, can be substantially different [108]. To avoid gradient accumulator overflow in (5.12), the precision of $\Delta_{W,n}$ (B_Δ) is bounded by:

$$B_\Delta \geq B_X + \log_2 N \tag{5.14}$$

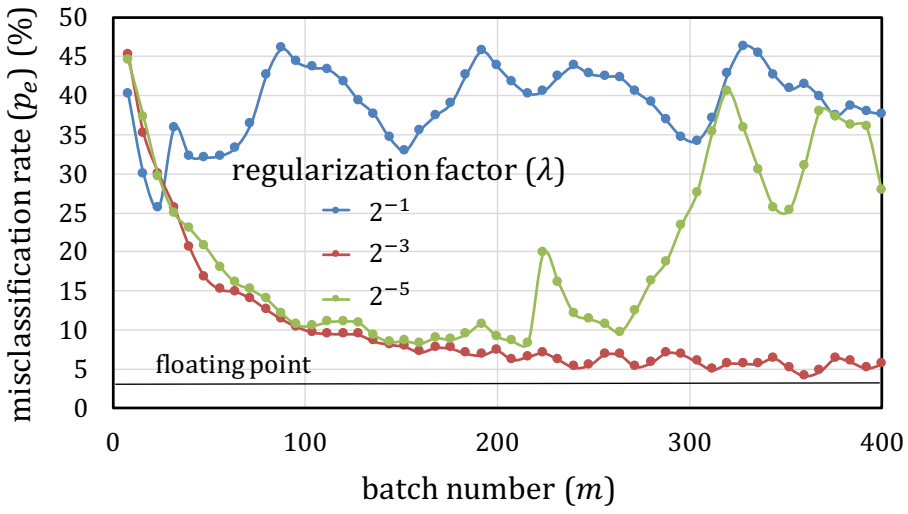
where B_X is the precision of X, which is fixed at 8-b in this application. We choose $B_\Delta = 16$ as $B_X = 8$ and we wish to accommodate batch sizes of up to $N = 256$.

It can be shown [108] that a necessary condition for convergence (stopping criterion) of the SGD update in (5.11) is given by:

$$B_{\text{WUD}} \geq 1 - \log_2 \gamma \tag{5.15}$$



(a)



(b)

Figure 5.7: Measured learning curves showing the impact of: (a) batch size N and learning rate γ with $\lambda = 2^{-4}$, and (b) regularization factor λ with $\gamma = 2^{-4}$ and $N = 64$.

We choose $B_{\mathrm{WUD}} = 16$ as algorithmic simulations indicate that the algorithm converges with a learning rate $\gamma \geq 2^{-15}$. Additionally, the batch-mode algorithm offers an interesting trade-off between the batch size N and the learning rate γ which is studied in Section 5.4.2.

The regularization factor λ has an optimum value that lies in between an upper bound that constrains the magnitude of W and a lower bound needed to avoid overflow in the weight accumulator (5.11). It can be shown that a sufficient condition to prevent overflow in the weight accumulator is given by:

$$\lambda \geq \Pr\{y(\mathbf{W}'^{T}_{\mathrm{opt}}\mathbf{X} + b) < 1\} \tag{5.16}$$

where $\mathbf{W}'_{\mathrm{opt}}$ are the optimal weights after full convergence. Similarly, there exists an upper bound on λ beyond which $|W|$ gets constrained so heavily that the MSBs are zero.

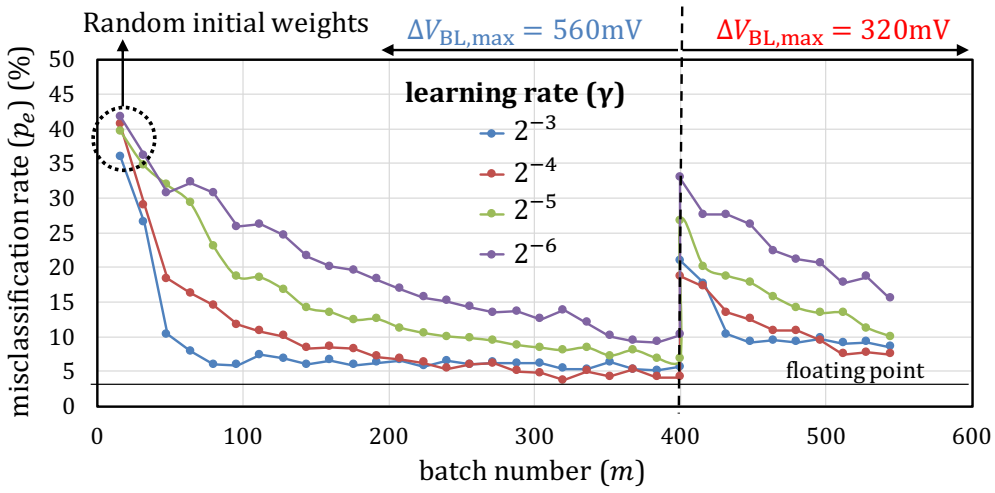
5.4 Experimental Results

This section describes the measured results from the prototype IC, and evaluates the effectiveness of on-chip learning in enhancing robustness. The 65 nm CMOS prototype IC (see Fig. 5.11) with a 16 kB SRAM is packaged in a 80-pin QFN. The area overhead of the in-memory computation circuits and the trainer is 15% and 35% of the IM-CORE area, respectively. The overhead of the trainer stems from the need to store intermediate gradients $\Delta_{W,n}$.

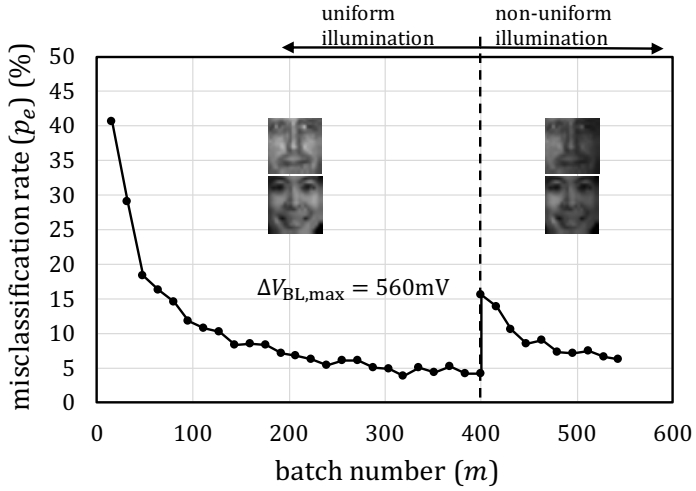
5.4.1 Training Procedure

The prototype IC is evaluated on the MIT CBCL face detection dataset [109]. The task is a binary classification problem with the dataset consisting separate sets of 4000 training and 858 test images with an equal number of 'face' and 'non-face' images. The input images were scaled down in size to 11×11 and then extended to accommodate the bias term b so that an 8-b 128-dimensional weight vector can be stored in four rows of the BCA.

During training, the batch $\{\mathbf{X}_n^{(m)}\}_{n=0}^{N-1}$ is generated from the training set by random sampling with replacement. During convergence, at the end of every 8-th batch, the misclassification rate $p_e = \Pr\{y \neq \hat{y}\}$ is calculated over



(a)



(b)

Figure 5.8: Measured learning curves showing robustness to: (a) process variations, and (b) variations in input statistics with a batch size $N = 64$ and a regularization factor $\lambda = 2^{-4}$.

the entire test set using the batch weight vector $\mathbf{W}_m$.

5.4.2 On-chip Learning Behavior

Measured learning curves in Fig. 5.7(a) show that the convergence is faster at a higher learning rate γ for both $N = 32$ and $N = 64$. Additionally, the learning curves become smoother and converge to a lower p_e with larger batch sizes, e.g., for $\gamma = 2^{-4}$, the misclassification rate p_e at $m = 400$ is lower for the batch size $N = 64$ than for $N = 32$. Figure 5.7(b) shows that the fixed-point algorithm converges for a regularization factor $\lambda = 2^{-4}$ but diverges for values of $\lambda = 2^{-1}$ and $\lambda = 2^{-5}$. For $\lambda = 2^{-5}$, the weights initially converge and then diverge due to overflow, whereas for $\lambda = 2^{-1}$, the algorithm does not converge at all because the MSBs which are used in the SVM dot product (5.1) remain at zero.

5.4.3 Robustness

Figure 5.8 shows that on-chip learning converges with randomly set initial weights in the BCA. Furthermore, the learning curves converge to within 1% of floating-point accuracy with 400 batch updates for $\gamma = 2^{-3}$ and 2^{-4} with $\Delta V_{\mathrm{BL,max}} = 560\,\mathrm{mV}$. The misclassification rate p_e increases dramatically to 18% when $\Delta V_{\mathrm{BL,max}}$ is reduced to 320 mV at $m = 400$. As discussed in Section 5.3.A, this increase occurs due to the increased impact of spatial variations. Continued on-chip learning reduces p_e down to 8% for $\gamma = 2^{-4}$ and 2^{-3} within 150 additional batch updates. Similar results are observed when the illumination of the input images changes abruptly at $m = 400$ (see Fig. 5.8(b)), where p_e increases to 16% and eventually reduces to 6% with further training. The measurements in Fig. 5.8 indicate the effectiveness of on-chip learning in enhancing the robustness to variations in the process parameters and the input data statistics.

The chip-specific nature of these learned weights can be seen in Fig. 5.9 which shows the average p_e increases from 8.4% to 43% when weights learned on a different die are used. This result further highlights the need for on-chip learning.

The receiver operating curve (ROC) characterizes the true positive rate

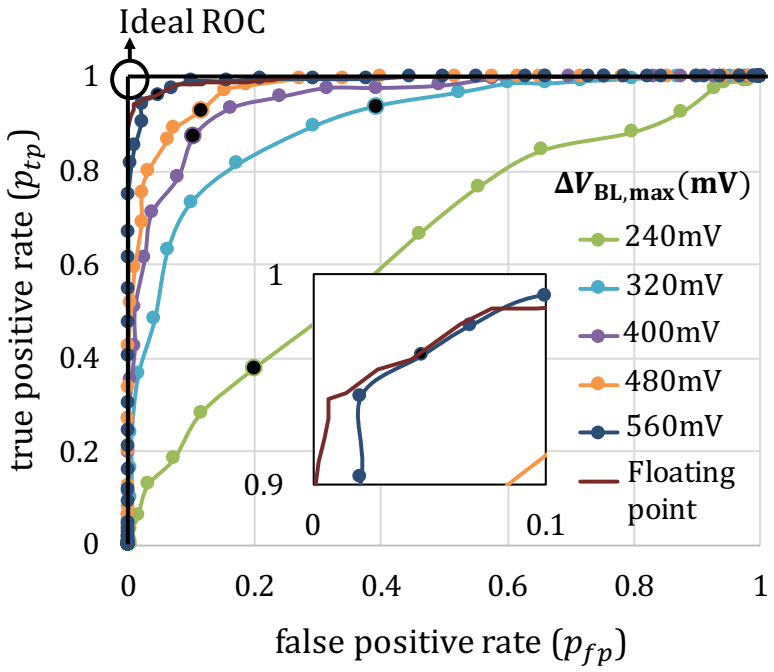
trained on \ tested on	Chip1	Chip2	Chip3	Chip4	Chip5
Chip1	8.25	38.3	48.3	51.5	48.8
Chip2	45.8	9	48	49.8	34.5
Chip3	47	51.3	8.5	29.8	49.3
Chip4	51.5	51	17.5	8.25	51.3
Chip5	38.3	18	48.5	48.5	8

Figure 5.9: Misclassification rate (p_e) measured across chips when the weights trained on a different die are used.

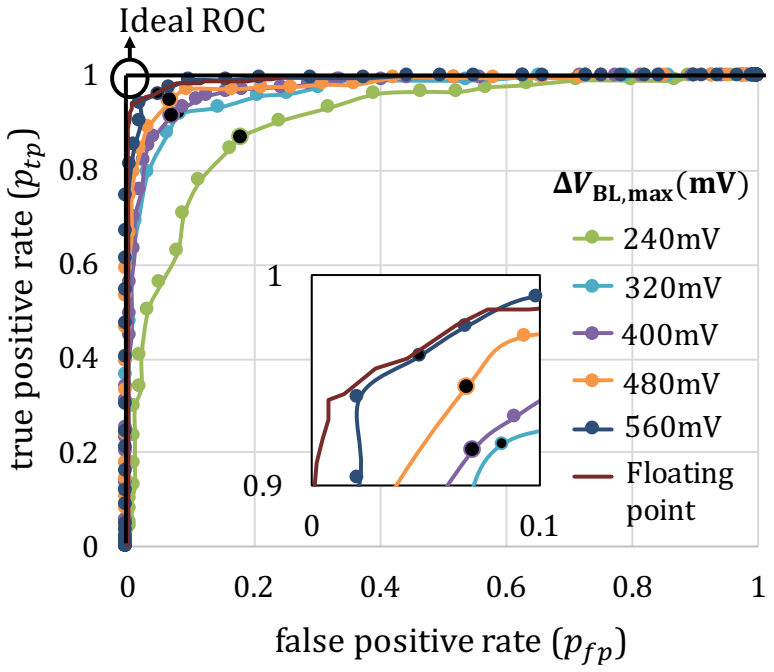
p_{tp} (the fraction of positive examples (faces) classified correctly) with respect to false positive rate p_{fp} (the fraction of negative examples (non-faces) classified as positive). The false positive rate p_{fp} was set by digitally adjusting the ADC output offset, and p_{tp} and p_{fp} were measured over the entire test dataset. With off-chip trained weights, the ROC of the classifier (Fig. 5.10(a)) degrades (moves away from the ideal) with decreasing $\Delta V_{\text{BL,max}}$. Additionally, the default operating point, i.e., without ADC offset cancellation (black markers in Fig. 5.10(a)), are, away from the optimal (knee of the ROC). However, with on-chip training (see Fig. 5.10(b)) the ROCs shift towards the ideal (upper left corner), and the default operating point (black markers) also moves automatically to the optimal location (the knee), thereby eliminating the need for offset tuning.

5.4.4 Energy Consumption

The minimum $\Delta V_{\text{BL,max}}$ required to achieve a misclassification rate $p_e \leq 8\%$ without compensating for process variations is 520 mV (see Fig. 5.12(a)). On-chip training enables the IC to achieve a misclassification rate below 8% at a 38% lower $\Delta V_{\text{BL,max}} = 320\,\text{mV}$. Operating with $\Delta V_{\text{BL,max}} = 320\,\text{mV}$ also enables the IC to operate with a lower $V_{DD,\text{IM-CORE}} = 0.675\,\text{V}$ without destructive reads as compared to operating at $V_{DD,\text{IM-CORE}} = 0.875\,\text{V}$ when $\Delta V_{\text{BL,max}} = 520\,\text{mV}$ (see Fig. 5.12(b)). Thus, on-chip learning enables the

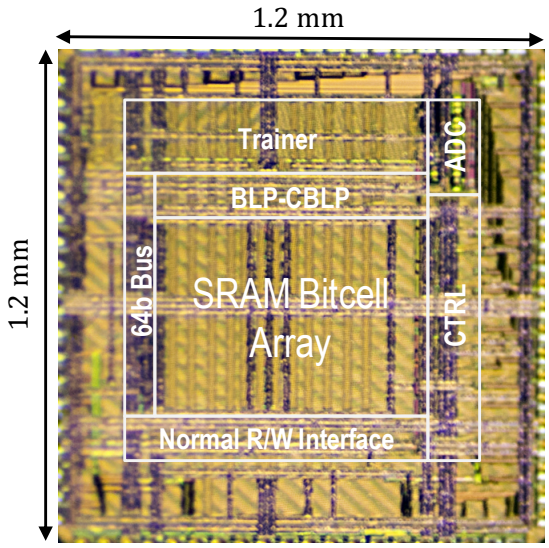


(a)



(b)

Figure 5.10: Receiver operating curves (ROC) measured with: (a) off-chip trained weights, and (b) on-chip trained weights. Black markers represent default operating points.



Technology	65nm CMOS	
Die size	1.2 mm × 1.2mm	
Memory capacity	16KB (512 × 256)	
Nominal supply	1.0 V	
CTRL operating frequency	1 GHz	
Energy per decision (nJ)	Test	0.21
	Training	0.34
Average throughput (decision/s)	Test	32.3 M
	Training	21 M

Figure 5.11: Die photograph and chip summary.

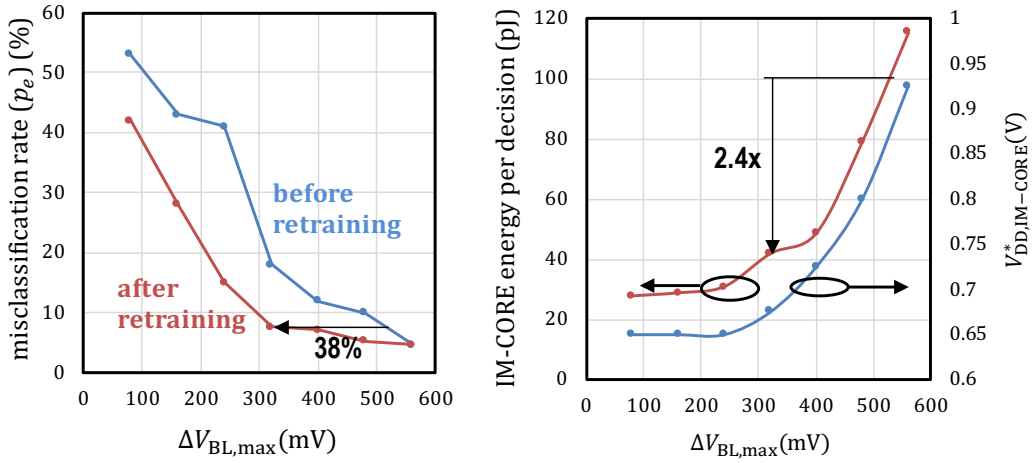


Figure 5.12: Measured misclassification rate p_e showing 38% reduction in BL swing attributed to on-chip learning, leading to a 2.4× reduction in energy ($V^*_{DD,\text{IM-CORE}}$ is the minimum IM-CORE supply to avoid destructive reads).

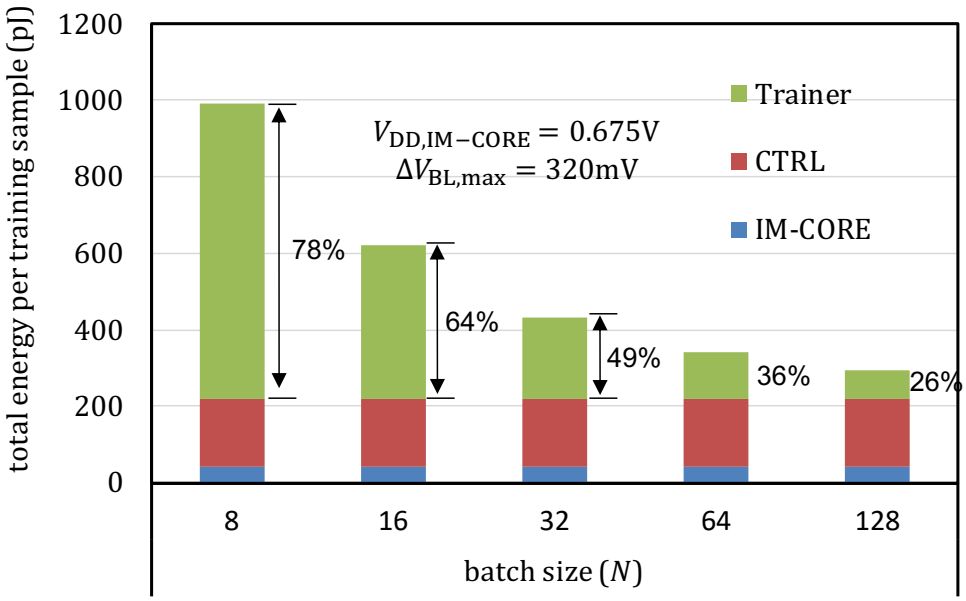


Figure 5.13: Energy overhead of on-chip training with respect to batch size.

reduction in IM-CORE energy by 2.4× at iso-accuracy. This energy gain ranges from 1.5×-to-2.4× for p_e in the range 5%-to-8%.

The energy cost of training is dominated by SRAM writes of the updated weights in (5.11) at the end of each batch (see Fig. 5.13). This cost reduces with increasing batch size N, reaching 26% of the total energy cost, for a $N = 128$. At $N = 128$, 60% of the total energy can be attributed to CTRL, whose energy reduces with increasing BCA size.

Figure 5.14 shows the energy breakdown of the prototype IC compared to a conventional digital reference architecture (CONV). CONV is a 2-stage pipelined architecture comprising an SRAM of the same size as in the prototype IC, and a synthesized digital block. The conventional SRAM has a column multiplexer ratio of $4:1$, therefore requiring 16× more read accesses than the DIMA. The energy and delay numbers of CONV were based on measured read energy from the prototype IC and computational energy from post-layout simulations.

The energy and delay benefits of the prototype IC stem from: (a) simultaneously reading multiple rows and processing them in low swing analog domain, (b) eliminating the 4:1 column mux, and c) by aggressively reducing BL swing enabled by the use of chip-specific weights obtained via on-chip learning. When operating with pre-trained weights at $\Delta V_{\mathrm{BL}} = 560\,\mathrm{mV}$, the

prototype IC shows a 7.8× reduction in energy compared to CONV, where the contributions due to (a) and (b) are estimated to be 5× and 1.6×, respectively. Use of chip-specific weights via on-chip learning increases the energy reduction to 21×. Along with the reduction in energy, the prototype IC simultaneously shows an overall 4.7× reduction in delay, thereby achieving an overall 100× reduction in EDP during inference. Due to the use of digital read and write operations during the weight update, the energy gain during training reduces to 6.2× (from 21× during inference) at a batch size $N = 64$.

Table 5.1 compares the prototype IC to other in-memory dot product implementations. The prototype IC operates with the energy-efficiency of 42 pJ/decision at a throughput of 32 M decisions/s, which corresponds to a computational energy-efficiency of 3.12 TOPS/W (1 OP = one 8-b×8-b MAC) for inference. DIMA's energy-latency benefits arise primarily from reduced memory access costs, which tend to dominate with large SRAM bank sizes. Furthermore, DIMA is best suited for algorithms that that suffer most from memory access costs such as fully connected deep neural networks (FC-DNN). Therefore, we compare with [11] which implements a FC-DNN, and employs aggressive voltage/frequency scaling along with digital error compensation techniques such as RAZOR [110]. Employing reported arithmetic efficiency of 0.09-0.16 TOPS/W [11] and accounting for the difference in the process node (28 nm FD-SOI [11] vs. 65 nm bulk CMOS), we find that the prototype IC achieves a 30× savings in energy accompanied by a 1.8× savings in delay to implement an 8-b 128-wide dot product. These energy savings demonstrate the suitability of the proposed architecture for energy-constrained sensory IoT applications.

5.5 Conclusions

This chapter has presented an IC realization of a deep in-memory classifier for the SVM algorithm with on-chip learning in a 65 nm CMOS process. On-chip training overcomes the impact of variations in both process parameters and data statistics, thereby enabling the IC to operate at lower BL discharges than otherwise possible, thus saving energy.

In this chapter, we demonstrate that on-chip learning is effective in compensating for process variations in DIMA. However, to take full advantage

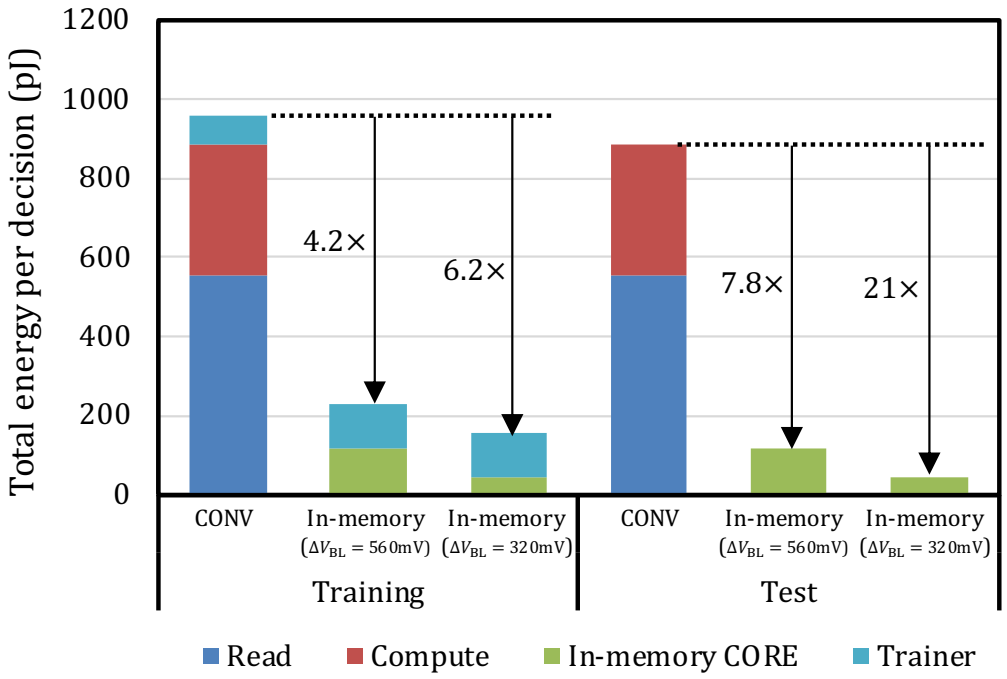


Figure 5.14: Measured on-chip energy for training (at $N = 64$) and inference compared to a conventional digital reference architecture (CONV), showing 21× reduction in energy consumption with a simultaneous 4.7× reduction in delay, leading to a 100× reduction in EDP during inference. The supply $V_{DD,\text{IM-CORE}}$ is 1 V and 675 mV when operating with a ΔV_{BL} of 560 mV and 320 mV, respectively.

Table 5.1: Comparison with prior in-memory dot product implementations.

	[30]	[28]	[29]	[32]	This work
Technology	130 nm	65 nm	65 nm	65 nm	65 nm
Algorithm	AdaBoost	SVM	CNN	DNN	SVM
Dataset	MNIST	MIT-CBCL	MNIST	MNIST	MIT-CBCL
On-chip memory (kB)	2	16	2	0.5	16
Bit-cell type	6T	6T	10T	6T[1]	6T
Energy/ decision (pJ)	600	400	-	-	42
Decisions/s	7.9M	9.2M	-	-	32M
Precision[2] ($B_X \times B_W$)	5×1[s]	8×8	7×1[s]	1×1	8×8[s]
Efficiency[3] (TOPS/W)	350	1.25	14	55.8	3.125 (1.07)[4]
Throughput[3] (GOPS)	819	4.17	5.35	1780	4.13
E_{MAC}[5] (pJ)	0.003	0.8	0.071	0.018	0.32 (0.92)[4]
$E_{\mathrm{MAC},p}$[6] (fJ)	0.56	12.5	10.2	17.9	4.9 (14.5) [4]

[1] separate left and right word lines (WLs) used

[2] signed number indicated by [s]

[3] 1 OP = 1 Multiply and accumulate

[4] at $\Delta V_{\mathrm{BL}} = 320\,\mathrm{mV}(560\,\mathrm{mV})$

[5] E_{MAC} is the energy of a single MAC operation.

[6] precision-scaled MAC energy $E_{\mathrm{MAC},p} = E_{\mathrm{MAC}}/(B_X \times B_W)$.

of this technique, a number of algorithmic, architectural, and circuit challenges need to be overcome. Unsupervised and semi-supervised learning algorithms are needed to avoid having to store the training set on-chip. Efficient write-back techniques would be required in order to minimize the energy consumption during training, especially for always-on systems that need to continuously track their environment. Realizing on-chip training is made much more challenging for large-scale deep networks. However, since both inference and trainer computations are based on matrix-vector operations, there is significant potential for employing DIMA to realize both the forward and backward parts of the network. In such cases, the impact of PVT variations on the learning behavior and inference accuracy is interesting to study.

If the impact of variation on each cell can be estimated, encoding techniques can be used instead of on-chip learning. While estimating the variation in current for each cell in SRAM is not straightforward, it is possible to do so in memoristor crossbars with simple modification. Chapter 6 presents a data encoding technique that exactly addresses this.

CHAPTER 6

ENHANCING WRITE ERROR-TOLERANCE OF RERAM CROSSBAR ARRAYS

Chapter 3 and Chapter 5 focus on in-memory architectures that are implemented on CMOS technology, in particular static random access memories (SRAMs). Alternatively, architectures that employ non-volatile resistive memory-based crossbars such as [93,111,112] have also gained popularity due to the large storage density, and due to the ease of realizing matrix-vector multiplications (MVMs) on them. Despite their benefits, crossbar architectures are highly susceptible to device and circuit non-idealities. In particular, conductance variations that stem from spatial variations and cycle-to-cycle (C2C) variations (*write noise*) are crucial.

This chapter proposes the Single-Write In-memory Program-vErify (SWIPE) to address the impact of conductance variations due to spatial and C2C variations. SWIPE achieves accurate bitcell writes for in-memory computing applications via a single scan of the array, thereby being 5×-to-10× more energy and latency efficient than conventional program-verify methods [113]. In this chapter, we demonstrate the effectiveness of SWIPE in enhancing the accuracy of DNNs realized on ReRAM crossbars in the presence of conductance variations, that stem from both spatial and C2C variations.

6.1 Background and Related Works

In this section, we provide the necessary background. We consider a crossbar implementation of the following MVM:

$$\mathbf{y} = \mathbf{W}^{\mathsf{T}}\mathbf{x} \tag{6.1}$$

where $\mathbf{x} = [x_1, \ldots, x_N]^{\mathsf{T}}$ is a $N \times 1$ the input vector, $\mathbf{y} = [y_1, \ldots, y_M]^{\mathsf{T}}$ is the $M \times 1$ output vector, and $\mathbf{W}$ denotes a $M \times N$ weight matrix with weights w_{ij} at i-th row and j-th column. Without loss of generality, we assume $x_i \in [0, 1]$

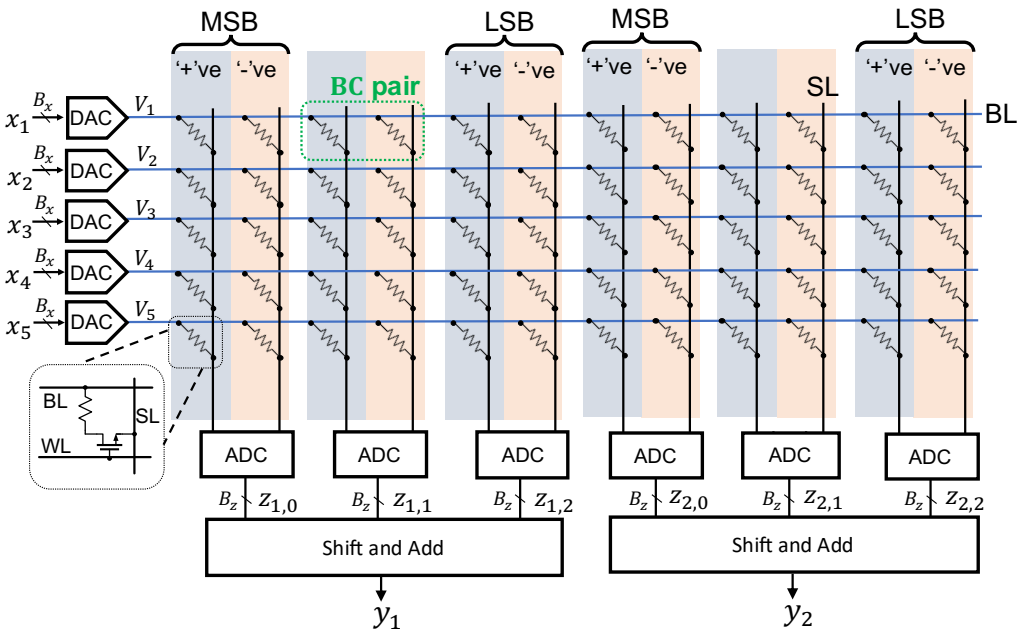


Figure 6.1: A 1T-1R resistive crossbar-based in-memory architecture realizing a signed multi-bit MVM computation via differential representation and bit-slicing, with parameters $M = 2,\ N = 5$, and $N_c = 3$.

and $w_{i,j} \in [-1, 1]$, with precisions B_x and B_w, respectively.

6.1.1 MVM via a Resistive Crossbar

The 1T-1R resistive crossbar-based in-memory architecture in Fig. 6.1 realizes a signed B_w-b×B_x-b MVM in (6.1). Each 1T-1R bitcell (BC) stores B_c bits and two adjacent BCs (BC pair) to realize a *signed* scalar using *differential representation* [111, 114]. Using a *bit-sliced* architecture [93, 112], a B_w-b signed weight $w_{i,j}$ is stored in N_c adjacent BC pairs with $B_w = N_c B_c + 1$. Thus, the $M \times N$ B_w-b weight matrix $\mathbf{W}$ requires a $2N \times N_c M$ crossbar.

The j^{th} partial dot product $z_{j,k}$ is computed in a pair of columns as follows:

$$z_{j,k} = \frac{\alpha}{\Delta G_{\max} V_{\max}} \sum_{i=1}^{N} \Delta G_{i,j,k} V_i \tag{6.2}$$

where $V_i = x_i V_{\max}$ is the voltage on the i-th bit-line (BL) (see Fig. 6.1), $\Delta G_{i,j,k}$ is the difference between conductances of BCs in the k-th BC pair associated with $w_{i,j}$, α is a constant, $\Delta G_{\max}$ is the conductance range, and

$V_{\max}$ is the voltage range for the DAC output V_i. Note that $\Delta G_{i,j,k} \in \mathcal{G} = \{g_{-L+1}, \ldots, g_0, \ldots, g_{L-1}\}$ where $|\mathcal{G}| = 2L-1$, $L = 2^{B_c}$, $g_l = l\Delta g$ ($l = -L+1, \ldots, L-1$), and Δg is the differential conductance step (see Fig. 6.2).

The partial dot products $z_{j,k}$ are digitized and summed after binary weighing to realize the final dot product y_j (see (6.1)) as follows:

$$y_j = \sum_{k=0}^{N_c-1} 2^{-kB_c} z_{j,k} = \sum_{i=1}^{N} w_{i,j} x_i \tag{6.3}$$

Note that the $2N_c$ cells associated with each weight parameter $w_{i,j}$ can be stored across multiple banks [115].

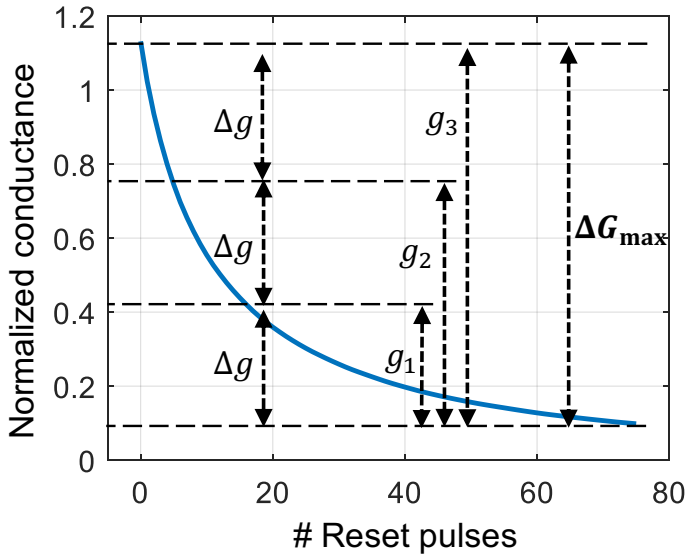
6.1.2 Non-idealities in Crossbar

ReRAM devices are programmed using two operations: (a) SET, and (b) RESET. The SET operation increases the conductance of the ReRAM device, while RESET reduces it. In most emerging non-volatile memory (eNVM) devices, the SET operation is abrupt, and only the RESET operation is used for multi-level conductance tuning [117]. To obtain the desired conductance change, RESET pulses are either modulated in time, amplitude, or the number of pulses [118]. ReRAM conductance as a function of the number of RESET pulses can be non-linear; appropriately choosing the number of pulses is required to achieve the desired conductance change (see Fig. 6.2(a)).

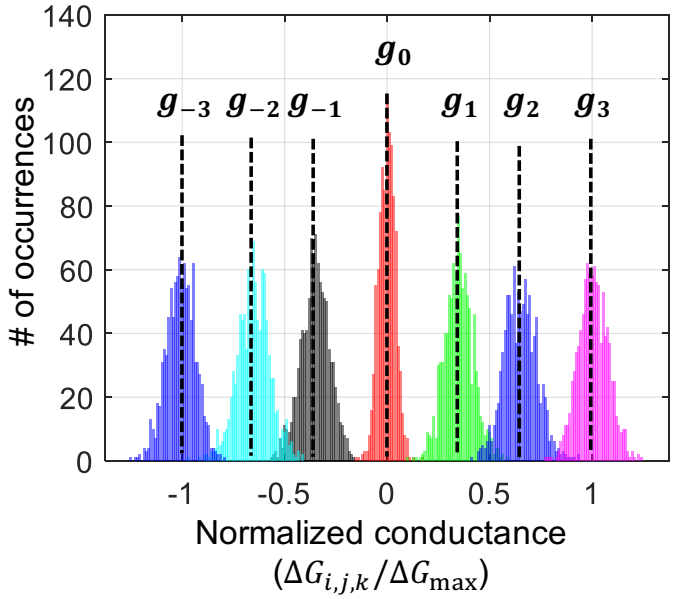
Stochastic non-idealities in a crossbar are due to: (a) *spatial variations* in the BCs, and (b) *write noise*, which includes C2C variations in the cell conductances. The C2C variations occur due to the unpredictability in the ReRAM conductance during the RESET or SET operation. For a BC pair, we model conductance variations due to C2C and the device mismatch as follows:

$$\Delta G_{i,j,k} = \Delta \tilde{G}_{i,j,k} + \eta_{i,j,k} \tag{6.4}$$

where $\Delta \tilde{G}_{i,j,k} \in \mathcal{G}$ the desired (ideal) cell conductance (*conductance state*), and $\eta_{i,j,k}$ is the variation in conductance caused by both spatial variations and write noise. The distribution of $\eta_{i,j,k}$ conditioned on the value of $\Delta \tilde{G}_{i,j,k}$ is denoted by $P_\eta(\eta_{i,j,k} | \Delta \tilde{G}_{i,j,k} = g_l) = \mathcal{N}(0, \sigma^2_{g,l})$ (see Fig. 6.2(b)). Additionally, the *read noise* during MVM computation, that includes thermal noise and shot noise, can be modeled as an additive Gaussian random variable sampled



(a)



(b)

Figure 6.2: ReRAM write noise with $B_c = 2$ bits/cell using the Stanford ReRAM Verilog-A model [116]: (a) normalized conductance vs. number of RESET pulses under ideal (no conductance variation) conditions. The four conductance states are obtained by equipartitioning the conductance range $\Delta G_{\max}$ into three steps, and (b) Monte Carlo simulations showing conductance variations on the conductance difference of the BC pair due to C2C variation and device mismatch when an average number of RESET pulses from (a) is applied to each cell in the BC pair.

in every read iteration as suggested in [114].

6.1.3 Related Works

The achievable inference accuracy of crossbar architectures is limited by IR drop, device non-linearity, thermal noise, process variations, stuck-at-faults, write noise, and limited device endurance. A number of works have addressed these challenges such as [114] (IR drop), [119] (device variations), and [120] (conductance variations). Recently, on-chip training methods have been proposed [121] to minimize the impact of die-specific variations.

Device variability in crossbar memories stems primarily from spatial variations and cycle-to-cycle (C2C) variations (*write noise*). The impact of spatial variations is die-specific and can be compensated for post-fabrication via on-chip learning [121] methods. However, these methods are expensive and can only be employed if the memory array is seldom rewritten into, and they do not address C2C variations which occur in every write cycle. Noise injection (NI) based one-time offline training methods [114, 119, 120] determine an averaged set of network parameters for an ensemble of dies, thereby avoiding the cost of on-chip learning. However, such methods incur a significant loss in the inference accuracy as compared to on-chip training methods.

Program-verify methods [113, 117, 122, 123] to address C2C variations are popular due to their high write accuracy. However, these techniques require multiple read and program iterations each time the crossbar array is written into. Furthermore, such methods can write only a single bitcell of a crossbar at a time [113], thereby incurring a high energy and latency overhead. These overheads are problematic when DNN accelerators employ crossbars with insufficient on-chip memory capacity [93, 111], thereby requiring frequent writes. Thus, there is a need for techniques that achieve the write accuracy of program-verify methods while minimizing their energy and latency overheads.

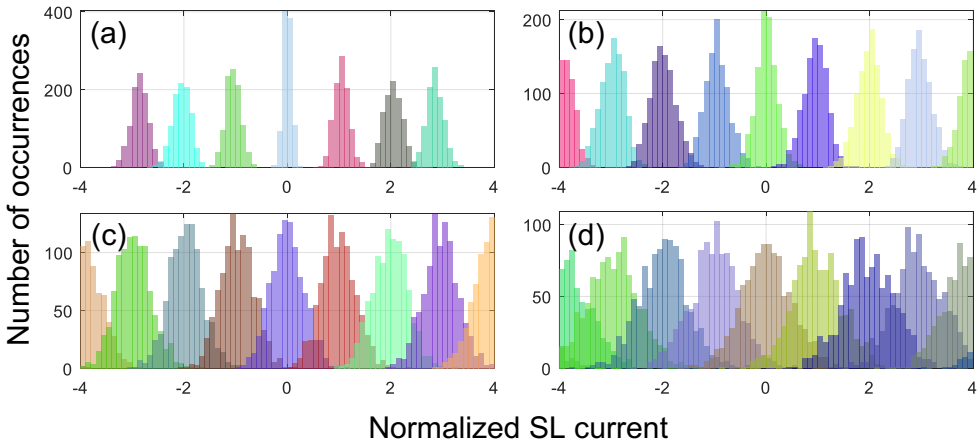


Figure 6.3: Histograms of the normalized SL current difference ($\propto z_{k,j}$) for different values of weights in the BCs where the average conductance standard deviation $\sigma_g/\Delta G_{\max} = 2.6\%$, $B_x = 1$, $B_c = 2$ with: (a) $N = 1$, (b) $N = 2$, (c) $N = 4$, and (d) $N = 8$. Stanford ReRAM Verilog-A model [116] was used for simulations.

6.2 The Single-Write In-memory Program-vErify (SWIPE) Method

In this section, we present the proposed Single-Write In-memory Program-vErify (SWIPE) to minimize the impact of write noise and device mismatch on the SNR of MVM computations in crossbar arrays.

6.2.1 Impact of Conductance Variations

During a crossbar-based in-memory MVM computation, individual BC currents are accumulated in the source lines (SLs) (see Fig. 6.1) to compute the dot product (6.2). This leads to an aggregation of noise due to device variations as shown below:

$$z_{j,k} + \gamma_{j,k} = \frac{\alpha}{\Delta G_{\max} V_{\max}} \sum_{i=1}^{N} \left(\Delta \tilde{G}_{i,j,k} + \eta_{i,j,k}\right) V_i \tag{6.5}$$

where $z_{j,k}$ the partial dot products from (6.2), and $\gamma_{j,k}$ is the total noise accumulated in the SL current. Monte Carlo simulations in Fig. 6.3 show that the outputs $z_{j,k}$ can be easily discriminated if $N = 1$ but becomes increasingly difficult as N increases due to the corresponding increase in the

variance of $\gamma_{j,k}s$.

The SNR of the final dot product output y_j in (6.3) is dominated by variations in cells with higher significance ($k = 0$), i.e., SNR of $z_{j,0}$. One way to overcome this SNR bound is to find a method to program $z_{j,k+1}$ such that it is inversely correlated with $\gamma_{j,0}, \ldots, \gamma_{j,k}$, i.e., the value stored in the less significant BC pair is chosen to compensate for the noise in BC pairs of higher significance. The proposed SWIPE method precisely does this by exploiting the bit-sliced nature of the crossbar-based in-memory architecture.

6.2.2 Proposed SWIPE Algorithm

The proposed SWIPE method writes the N_c BC pairs that store the word $w_{i,j}$ sequentially one BC pair at a time, in a specific order, i.e., from the most significant to the least significant BC pair. While writing the k-th BC pair of a given word $w_{i,j}$, SWIPE leverages the knowledge of the *composite* conductance value of the already written BC pairs 0 to $k-1$ (including their static variations) of that word. The knowledge and the parameters of distribution of device variations are leveraged to optimally choose the conductance value of the k-th BC pair to be written into. Note that BC pairs of all the words in a given row with the same significance are written in parallel, i.e., N_{col}/N_c BC pairs/row are programmed in every iteration. Furthermore, SWIPE programs each BC pair only once, making it significantly (5×-to-10×) more efficient compared to the standard program-verify techniques [113, 117, 122].

Formally described in Algorithm 2, each iteration k has following three steps (see Fig. 6.4(a)), where the cell indices i, j are dropped for simplicity:

Word-read: In this step, the composite value f_k of k already written BC pairs is read from the array (see Fig. 6.4(b)) as follows:

$$f_k = \frac{\alpha}{\Delta G_{\max}} \sum_{m=0}^{k-1} 2^{-mB_c} \Delta G_m \tag{6.6}$$

Note that, $f_k = 0$ for $k = 0$. Also, in absence of conductance variations, f_k is the $kB_c + 1$ bit quantized version of w, where w denotes the desired value of the word.

Estimate: In this step, f_k is used to determine the desired conductance

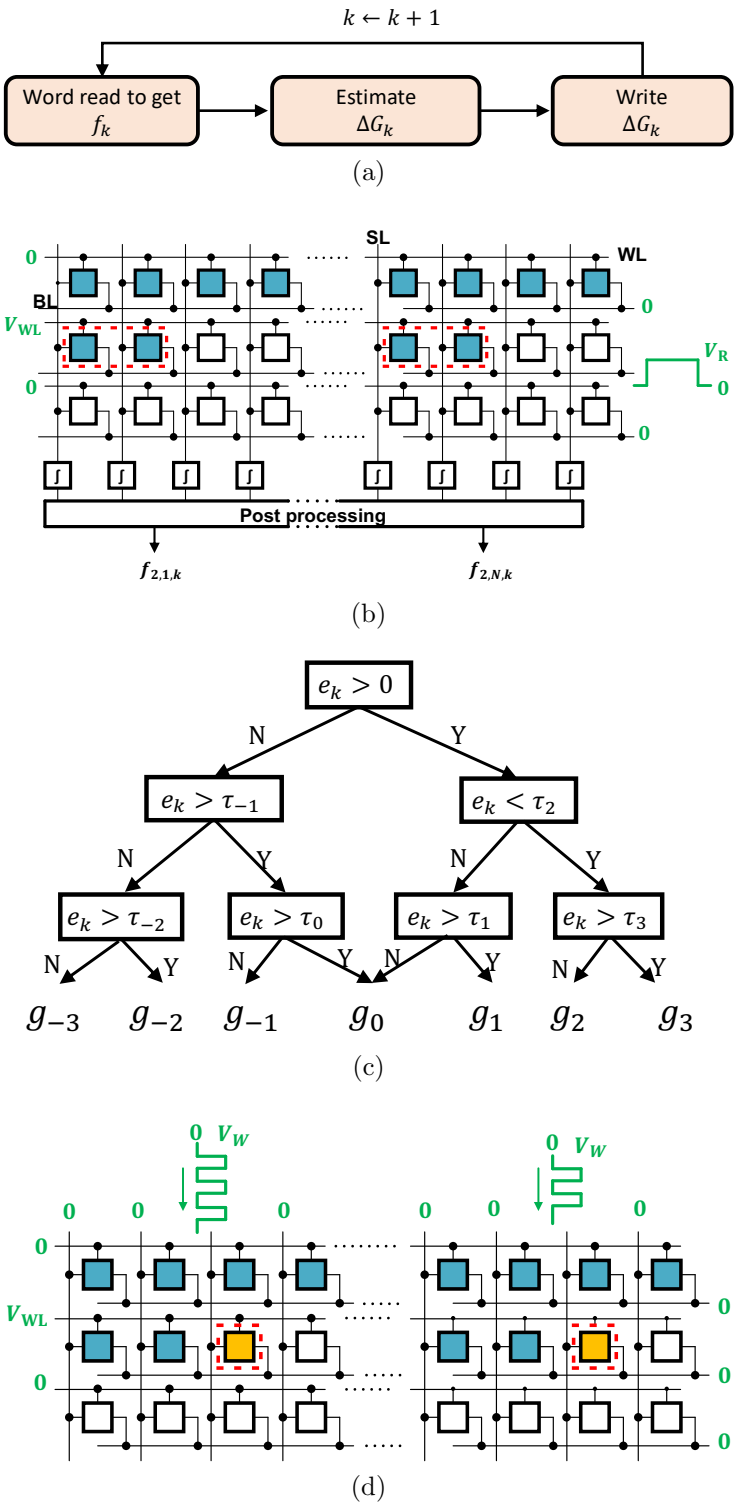


Figure 6.4: The proposed SWIPE method: (a) Flow diagram illustrating the three stages of SWIPE applied to the 2-nd row of a crossbar where $k = 2$, $N_c = 4$, and $B_c = 2$, (b) the word-read stage to obtain f_k, (c) decision tree to estimate ΔG, and (d) the write stage showing $1/N_c$-th row written.

state of the next BC pair $\Delta\tilde{G}_k$ (see Fig. 6.4(c)) as follows:

$$\Delta\tilde{G}_k = g_d \quad \text{if} \quad \tau_{d-1} \leq e_k < \tau_d \tag{6.7}$$

where the error $e_k = \alpha^{-1}\Delta G_{\max} 2^{kB_c}(w - f_k)$ quantifies how far f_k is from the desired value w, $g_d \in \mathcal{G}$, and the thresholds $\mathcal{T} = \{\tau_d\}_{d=-L+1}^{d=L-1}$ are predetermined based on the variances of write noise per conductance state.

Write: The estimated conductance state $\Delta\tilde{G}_k$ is written to the k-th BC pair, as shown in Fig. 6.4(d).

6.2.3 Choosing Optimal Thresholds

The thresholds $\mathcal{T}\{\tau_d\}_{d=-L+1}^{d=L-1}$ need to be chosen to maximize SNR. At iteration k, given that f_k, the optimal value for $\Delta\tilde{G}_k$ is obtained by solving:

$$\begin{aligned}
\Delta\tilde{G}_k &= \arg\min_{g_l \in \mathcal{G}} \mathbb{E}\big[(w - f_{k+1})^2 | f_k\big] \\
&= \arg\min_{g_l \in \mathcal{G}} \mathbb{E}\big[(e_k - g_l + \eta_{g,l})^2 | f_k\big] \\
&= \arg\min_{g_l \in \mathcal{G}} \big[(e_k - g_l)^2 + \sigma_{g,l}^2\big].
\end{aligned} \tag{6.8}$$

where $\eta_{g,l}$ is a random variable capturing the conductance variations when the conductance state is g_l. Given the conductance variances $\{\sigma_{g,l}^2\}$ (6.4), the optimal thresholds $\mathcal{T}_{\text{opt}}$ are computed by solving (6.8) for every value of e_k. In practice, $\mathcal{T}_{\text{opt}}$ are pre-computed once and stored on-chip, therefore only (6.7) needs to be implemented on-chip to find $\Delta\tilde{G}_k$.

6.2.4 Hardware Considerations

Implementing SWIPE in hardware requires minimal overhead since it can leverage the peripheral circuitry associated with existing crossbar based accelerators [93, 112].

Word-read: The *Word-read* operation to obtain f_k can be realized by employing one-hot encoded inputs during MVM (see Fig. 6.4(b)). However, the SL current needs to be amplified to match the ADC input dynamic range since one BC pair contributes to the SL current as compared to the standard MVM mode.

Algorithm 2 Single-Write In-memory Program-vErify (SWIPE).

Input: The target weight matrix $\mathbf{W}$, $\mathcal{G}$, $\{\sigma^2_{g,l}\}$, B_c, N_c.
Output: The crossbar array conductance $\Delta G_{i,j,k} \quad \forall \quad i,j,k$

Estimate $\{\tau_i\}_{i=-L+1}^{L-1}$ by solving (6.8)
for $i := 1$ to N **do**
  parallel for $j := 1$ to M **do**
    for $k := 0$ to $N_c - 1$ **do**
      Word-read row i to obtain $f_{i,j,k}$
      $e_{i,j,k} \leftarrow \alpha^{-1}\Delta G_{\max} 2^{kB_c}(w_{i,j} - f_{i,j,k})$
      Obtain $\Delta\tilde{G}_{i,j,k}$ as per (6.7)
      Write back $\Delta\tilde{G}_{i,j,k}$
    end for
  end parallel for
end for

Estimate: The threshold operation (6.7) requires one subtraction and B_c comparisons for each weight in a row (see Fig. 6.4(c)).

Write: The write operation in SWIPE requires N_c reads and write operations per row which makes it significantly efficient compared to conventional program-verify methods.

6.3 Simulation Results

In this section, we discuss the effectiveness of SWIPE in enhancing robustness to write noise and device mismatches. First, we describe a custom simulation methodology used to reflect the circuit and devise non-idealities in the system-level performance of the ReRAM crossbar. We study SWIPE in the context of DNN implementation on crossbars.

6.3.1 End-to-end Simulation Methodology

Figure 6.5 shows the evaluation methodology employed to quantify the system-level performance of DNNs on crossbars that integrate the circuit, architecture, and algorithmic parameters and design variables. We used the Stanford ReRAM Verilog-A model [116] for the ReRAM device characteristics, and commercial 22 nm FDSOI process to implement the access transistors. The design parameters used in the simulations are summarized in Table 6.1.

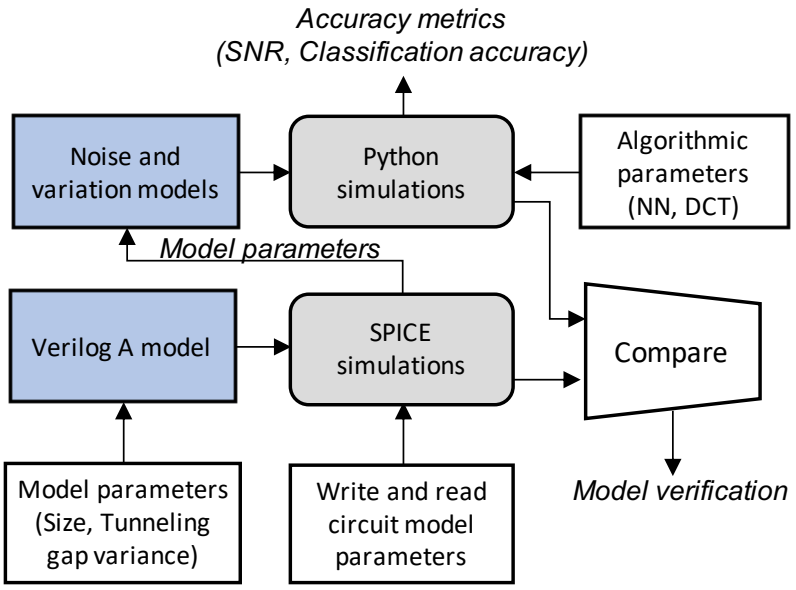


Figure 6.5: Evaluation methodology.

Table 6.1: Device and circuit parameters used in simulations.

Parameter	Value	Parameter	Value
$G_{\max}$ (μS)	500	$G_{\min}$ (μS)	50
B_x (bits)	8	Crossbar Size	$64 \times 128N_c$
ADC prec. (bits)	7−to−10	B_w (bits)	2−to−9
SL Res. (Ω/Cell)	0.86	BL Res. (Ω/Cell)	0.47

ReRAM Variation Model:

The cycle-to-cycle variations during the SET and RESET operations on the ReRAM device are modeled by introducing variation to the tunneling gap growth rate in the Verilog-A model as suggested in [116]. We modeled the device conductance distribution via circuit simulations of the ReRAM device, as shown in Fig. 6.2(b). The device-to-device variations can be modeled by introducing variation in the size of the ReRAM device.

Crossbar array model:

We developed a Python model of the ReRAM crossbar that incorporates the effects of ReRAM device variations. We verified this model against circuit simulations of a 16×16 crossbar. For circuit simulations, we modeled ReRAM devices with equivalent resistors by appropriately choosing their conductance values. We used ideal voltage sources for inputs on BLs, and ideal OpAmp-

based current integrators on the SLs.

We first compared the MVM operations on Python and circuit model by encoding conductance values based on 100 randomly chosen matrices and input vectors. We find that the Python model is within 0.2% and 1.1% of the circuit model with and without SL and BL parasitic resistances, respectively. The outputs of the Python and circuit models were compared to ideal expected outputs to verify the consistency of these models. The SL currents were quantized in post-processing assuming an ideal ADC. Next, we verified the data encoding procedures in SWIPE with circuit simulations. For each RESET operation, we sampled the BC conductance from ReRAM noise models obtained via Monte Carlo simulations of the Verilog A model.

Application-level Simulations:

The Python model was used for large scale statistical simulations of the application-level performance of the crossbar architecture. For the DNN simulations, we embed this Python model of the crossbar array into the PyTorch framework. The neural networks were mapped onto the crossbar using a naive network partitioning techniques, such as those presented in [124]. In order to ensure that the network can be easily quantized, we ensure the weights to lie between $[-1, +1]$ by clipping and scaling the weights during training. We study the impact of write noise on application-level accuracy metrics by varying $\sigma_g/\Delta G_{\max}$ where σ_g^2 is the conductance variance averaged over all the conductance states as shown below:

$$\sigma_g^2 = \frac{1}{2L-1}\sum_l \sigma_{g,l}^2 \tag{6.9}$$

6.3.2 Optimality of SWIPE Thresholds

To demonstrate the optimality of the thresholds $\mathcal{T}$ obtained from (6.8), we observe the impact on the SNR of outputs y, SNR_y, under a skew s in each threshold $\tau \in \mathcal{T}$ where $\tau_{\text{skew}} = \tau_{\text{opt}} + s\Delta g$. Figure 6.6 shows that the the pre-ADC SNR of the output y_j (SNR_y) of 16-point discrete Fourier transforms (DFTs) is maximized when $s = 0$, $\tau_{\text{skew}} = \tau_{\text{opt}}$. Interestingly, when σ_g and B_c increases, the SNR_y remains flat with s as observed with $B_c = 3$ in Fig. 6.6(b). This is because conductance variations σ_g being much larger than the smallest quantization step, Δg, swamps out the effect of non-ideal

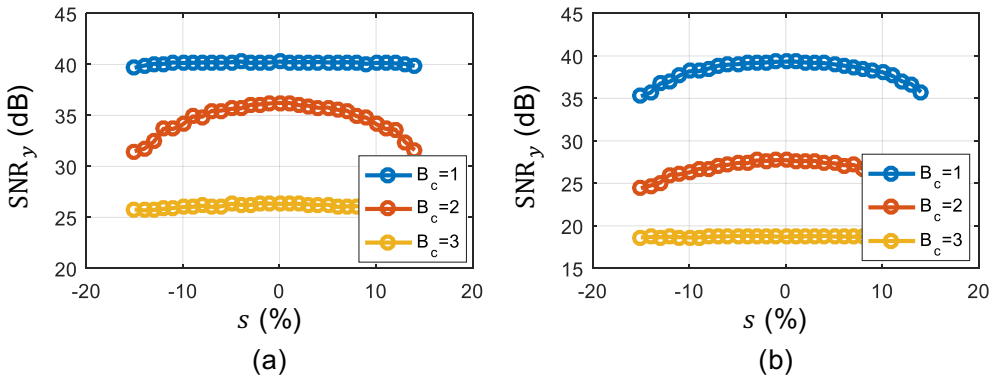


Figure 6.6: SNR$_y$ of 16-point DFT implementation on a $16 \times 32N_c$ crossbar with respect to the skew s in the thresholds with $B_w = 7$: (a) $\sigma_g/\Delta G_{\max} = 2.7\%$, and (b) $\sigma_g/\Delta G_{\max} = 10.8\%$. Input SNR was set at 41 dB.

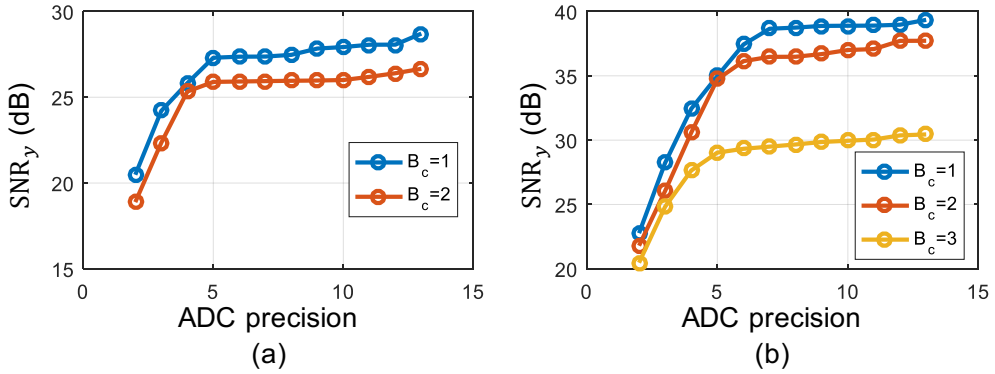
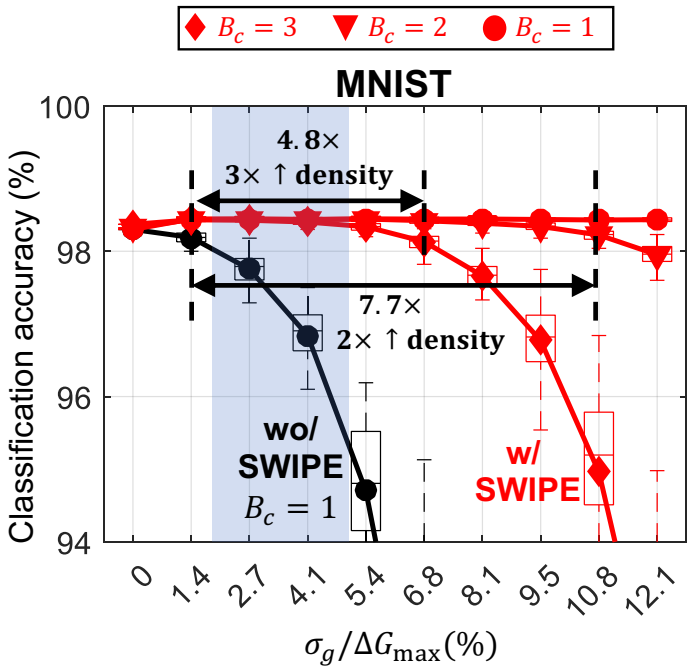


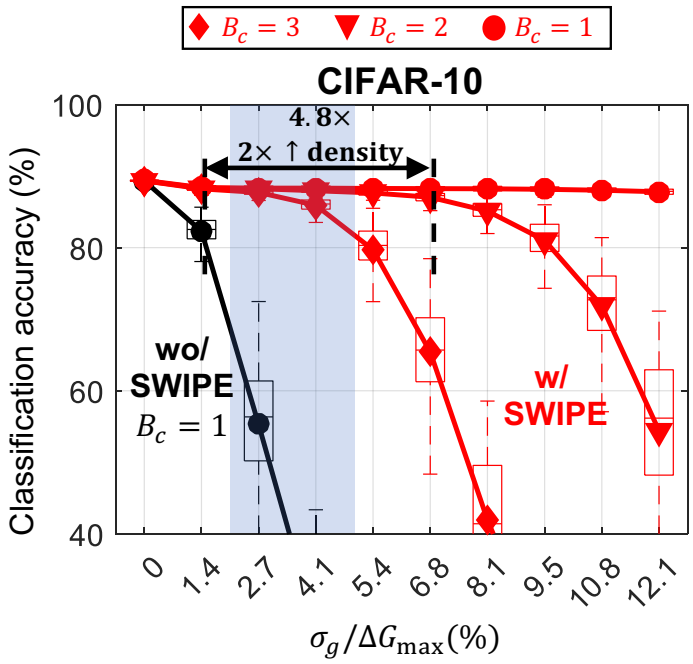
Figure 6.7: SNR$_y$ of 16-point DFT implementation on a $16 \times 32N_c$ crossbar with respect ADC precision during the word-read operation in SWIPE for: (a) $B_w = 5$, and (b) $B_w = 7$.

thresholds.

Since, in SWIPE, the accuracy of the threshold operation (see Fig. 6.4(c)) is critical to maximize SNR$_y$, therefore, the ADC precision during the word-read operation needs to be high enough for accurate thresholding. Figure 6.7 shows that the ADC precision should be at least B_w for SNR$_y$ to be within 3 dB of the maximum SNR$_y$. Note that the SNR improves for ADC precision higher than B_w as it enables a more accurate implementation of (6.7). Therefore, we choose ADC precision to be $B_w + 2$ in the rest of the chapter.



(a)



(b)

Figure 6.8: Accuracy in the presence of SWIPE with respect to average BL conductance variations for: (a) LeNet-300-100 on the MNIST dataset, and (b) the 8-layer CNN on the CIFAR-10 dataset. The box plots show the spread in network accuracy over 100 iterations. The shaded region marks the typical conductance variation range [91].

6.3.3 Robustness Improvements in DNN Implementations

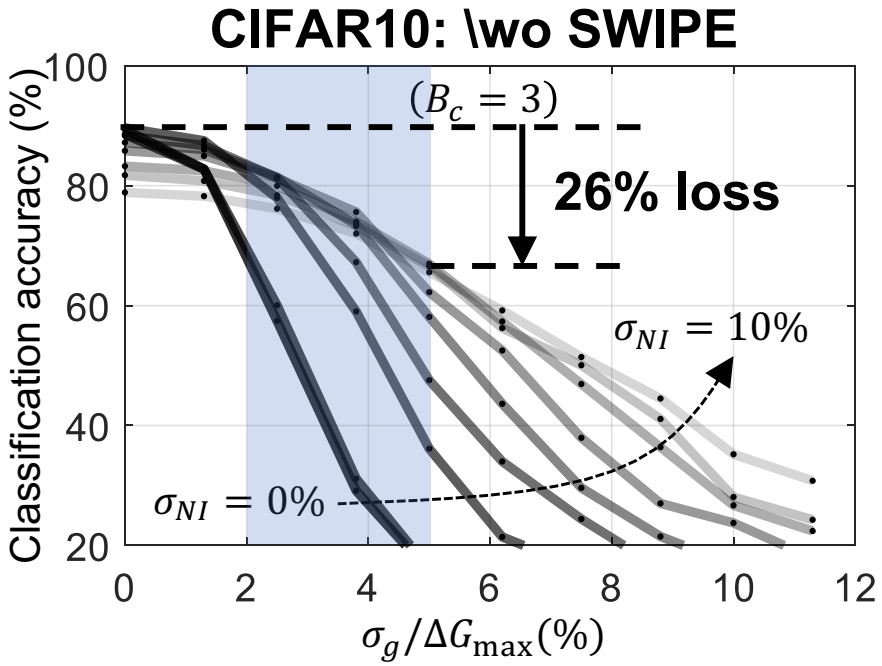
We study two DNN architectures: (a) LeNet-300-10, and (b) an 8-layer CNN with 7 convolution layers[1] followed by an AveragePool and a fully connected layer for classification on the MNIST and the CIFAR-10 datasets, respectively. Both networks employ Batch-Norm and a ClippedReLU nonlinearity at the output of each layer. We choose $B_w = 7$ to limit the loss in accuracy to $< 0.2\%$.

Figure 6.8 shows that SWIPE improves the robustness to conductance variations on both networks. The typical conductance variations in ReRAM due to write noise and device mismatch are about 2%-to-5% [91]. In this range, operating without SWIPE results in loss in accuracy with even with $B_c = 1$. In contrast, operating with SWIPE for $B_c = 1$ results *no loss in accuracy* in both networks for variations as high as $> 12\%$. These gains in robustness translates to gains in density, since SWIPE allows us to operate in the typical variation range with $< 1\%$ drop in accuracy for $B_c \leq 2$, and $B_c \leq 3$, on CIFAR-10 and MNIST, respectively. Thus, SWIPE allows us to simultaneously enhance robustness and density by 4.8×-to-7.7× and 2×-to-3×, respectively.

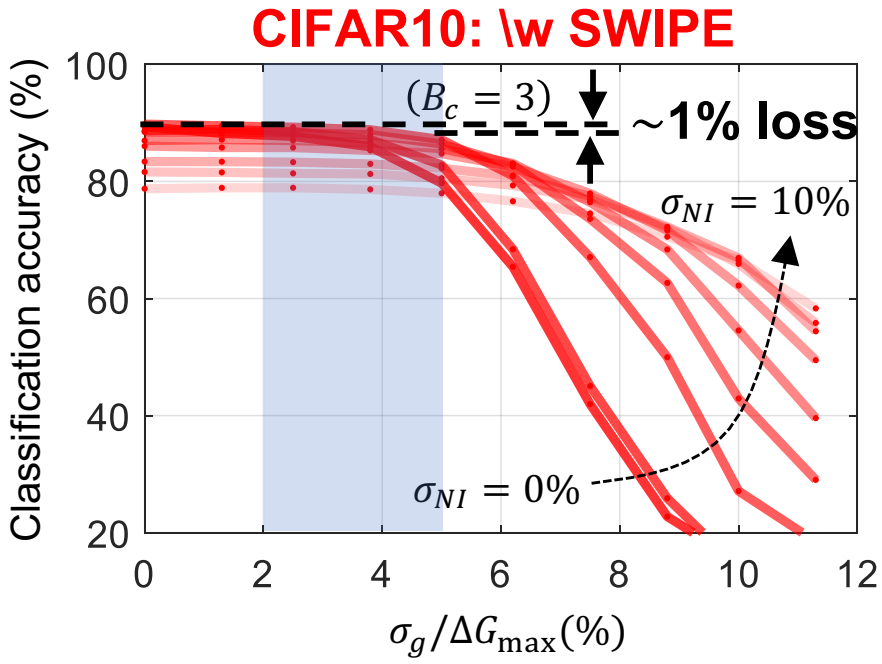
In NI-based training, Gaussian noise $\mathcal{N}(0, \sigma^2_{\text{NI}})$ is added to network weights during the feed-forward pass of the back-propagation iterations. NI-based training has been observed to improve robustness to line resistance [114], and to device mismatch and conductance variation in [119,120]. Though NI-based training improves robustness (see Fig. 6.9(a)), we observe that it degrades the maximum achievable accuracy at $\sigma_g/\Delta G_{\max} = 0$ and still results in 26% accuracy loss in the typical conductance variation range (shaded) (2%-to-5%) on the CIFAR-10 dataset. In contrast, Fig. 6.9(b) shows that augmenting SWIPE with NI-based training results in $< 1\%$ loss in accuracy with $B_c = 3$ within the typical conductance variation range.

Figure 6.10(a) shows that NI-based training improves robustness to read noise in the absence of write noise. However, in the presence of write noise, the classification decreases dramatically by 48% (see Fig. 6.10(b)) on the CIFAR-10 network. This loss in accuracy due to write noise is recovered by using SWIPE, as shown in Fig. 6.10(c). Thus, augmenting SWIPE with NI-

[1]The 7 convolutional layers are (3C64S1)×3−(3C128S2)−(3C128S1)−(3C256S2)−(3C512S1), where (aCbSc) indicates $a \times a$ kernel, b output channels, and stride c.



(a)



(b)

Figure 6.9: Accuracy of the 8-layer CNN for the CIFAR-10 dataset with NI-based trained where σ_{NI} ranges from 0% to 10% in steps of 1.25%, and with $B_c = 3$: (a) without SWIPE, and (b) with SWIPE. The shaded region marks the typical conductance variation range [91].

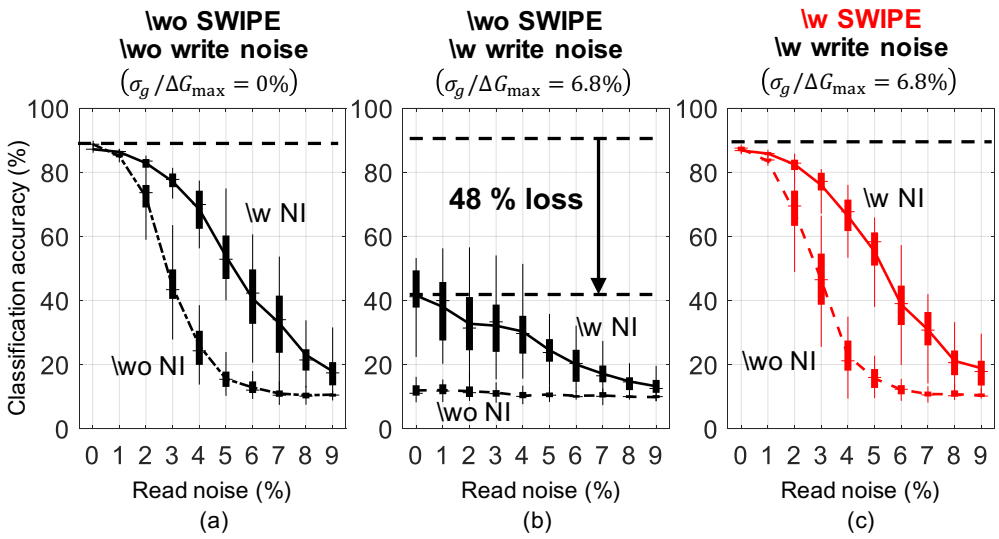


Figure 6.10: Accuracy of the 8-layer CNN on the CIFAR-10 dataset in the presence of read noise with $B_c = 2$: (a) without SWIPE and write noise ($\sigma_g/\Delta G_{\max} = 0\%$), (b) without SWIPE and with write noise ($\sigma_g/\Delta G_{\max} = 6.8\%$), and (c) with SWIPE and write noise ($\sigma_g/\Delta G_{\max} = 6.8\%$); $\sigma_{\text{NI}} = 5\%$ was used to train with NI. Box plots show the spread in accuracy over 100 iterations.

based training enables the design of crossbar-based in-memory architectures that are robust to both read and write noise.

6.4 Conclusions

This chapter presents SWIPE method to enable efficient and accurate writes in the presence of write noise and devise mismatch for in-memory crossbars. We demonstrate that SWIPE enables DNN implementation on ReRAM crossbars with $< 1\%$ loss under typical values of write noise and devise variations. Augmenting SWIPE with NI-based training enables DNN implementation on crossbars that are simultaneously robust to both write and read noise. The device variability requirements for storage are different and more relaxed than those of in-memory computing. Enabling in-memory architectures in emerging devices is therefore challenging and may require compensation techniques like those introduced in this chapter. Next, in Chapter 7 we present in-memory computing for NAND flash memories.

CHAPTER 7

DEEP IN-MEMORY ARCHITECTURES FOR NAND FLASH MEMORIES

Flash memories are widely employed in mobile devices and solid-state drives (SSD). NAND flash memories, in particular, have become an industry standard for large-scale storage. Since NAND flash memories suffer from I/O interface bandwidth and energy limitations, in-memory computing in NAND flash will have a massive impact. Chapter 3 studied in-memory architectures in CMOS technology, and Chapter 6 studied the impact of non-idealities in hybrid technologies such as ReRAM. However, in-memory computing using specialized memory technologies and processes such as NAND flash also brings many unique challenges. Challenges primarily arise from the stringent pitch matching constraints, large BL capacitors and the low mobility of the transistors in NAND flash memories.

This chapter[1] proposes deep in-memory architecture for NAND flash (DIMA-F) which brings computing functionality into storage class memories. DIMA-F reads the stored data and processes highly parallel dot products on single-level cell (SLC) NAND flash memories in the analog domain. DIMA-F is evaluated in the context of face detection and face recognition on the Caltech 101 database [126] and the Extended Yale B database [127], respectively. System-level simulations show marginal degradation in accuracy as compared to fixed point implementations, while achieving between 8×-to-23× energy savings, 9×-to-15× throughput gain, and 72×-to-345× improvements in energy delay product (EDP) compared to the conventional NAND flash architecture incorporating an external digital ASIC for computation.

[1]Adapted from: Sujan K. Gonugondla, Mingu Kang, Sean Eilert, Mark Helm, and Naresh R. Shanbhag, "Energy-Efficient Deep In-memory Architecture for NAND Flash Memories" *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2018, © IEEE. [125]

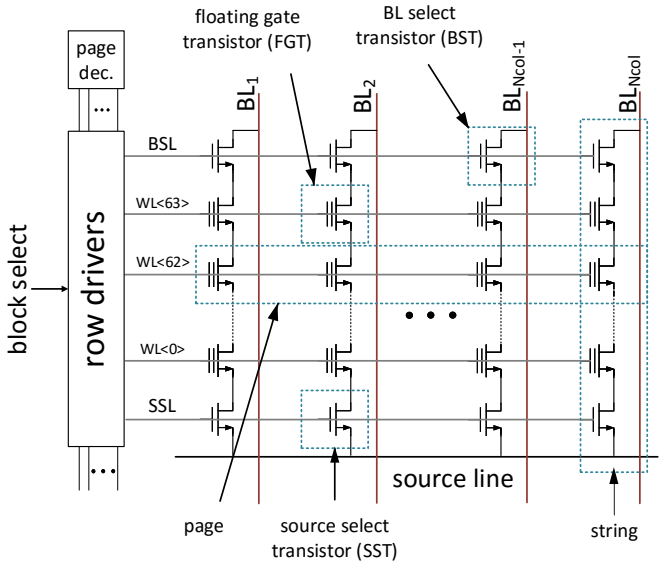
7.1 Background

This section provides the necessary background into NAND flash memories and its associated terminologies.

7.1.1 SLC NAND Flash Memory Architecture

NAND flash is a non-volatile storage/memory architecture that uses floating gate (FG) transistors as the basic storage cell. Data in NAND flash memories is stored as threshold voltages of the FG transistors which are induced by the charges on the FGs. In SLC NAND flash memory, the FG transistors have two states, i.e., an erased state (low threshold voltage) and a programmed state (high threshold voltage), corresponding to a single logical bit.

(a)



(b)

Figure 7.1: Architecture of a conventional SLC NAND flash: (a) a *plane*, and (b) a *block*.

A NAND flash chip contains a memory array, a control unit, high voltage generation circuitry for read and write operations, buffers to store/transmit data, and I/O interface circuitry. NAND flash memory is organized as multiple memory banks referred to as *planes*. Figure 7.1(a) shows the typical

architecture of a NAND flash plane. Each plane is further horizontally divided into *blocks*. A block is in turn divided into *pages* horizontally and *strings* vertically. NAND flash strings typically contain 64-to-128 FG transistors connected serially as shown in Fig. 7.1(b). They are accessed through a group of 64-to-128 word-lines. The data stored across the FG transistors sharing a single WL is called a page.

7.2 Deep In-Memory Architecture for NAND Flash (DIMA-F)

Though bringing computational functionality to NAND flash memory is highly beneficial in relaxing the constraints imposed by I/O circuitry, there are major challenges to be addressed. These challenges arise from small bit-cell pitch, high variability of NAND flash memories, and limitations in speed of NAND flash technologies for computations.

Figure 7.2 shows the proposed architecture for DIMA-F. It consists of the following blocks: (a) a memory array that allows *multi-column functional read* (MC-FR) that converts a W-b word to an analog voltage on a capacitor, (b) a *multi-bit-line processor* (MBLP) pitch-matched to BLs read in MC-FR that performs scalar multiplication, (c) a buffer that stores a reference vector or weights used during the MBLP operations, (d) a cross bit-line processor (CBLP) that performs dimensionality reduction by summation to implement dot product, and (e) an ADC or slicer that converts the analog output of the CBLP into the digital domain.

7.2.1 Multi-column Functional Read (MC-FR)

Consider a data word D stored as a W-b binary vector $\mathbf{d} = \{d_{W-1}, d_{W-2}.., d_0\}$. The goal of the MC-FR operation is to read in an analog voltage proportional to the decimal value of the data stored in the flash array ($\Sigma_{n=0}^{W-1} 2^n d_n$). To enable this the bits are stored horizontally in a page as shown in Fig. 7.3. Hence, N_{BL}/W words per plane are read in parallel using MC-FR, where N_{BL} is the number of BLs in a plane.

Figure 7.3 shows the architecture and Fig. 7.4 shows the timing for MC-FR. During MC-FR, the WL associated with the page being read is set to a

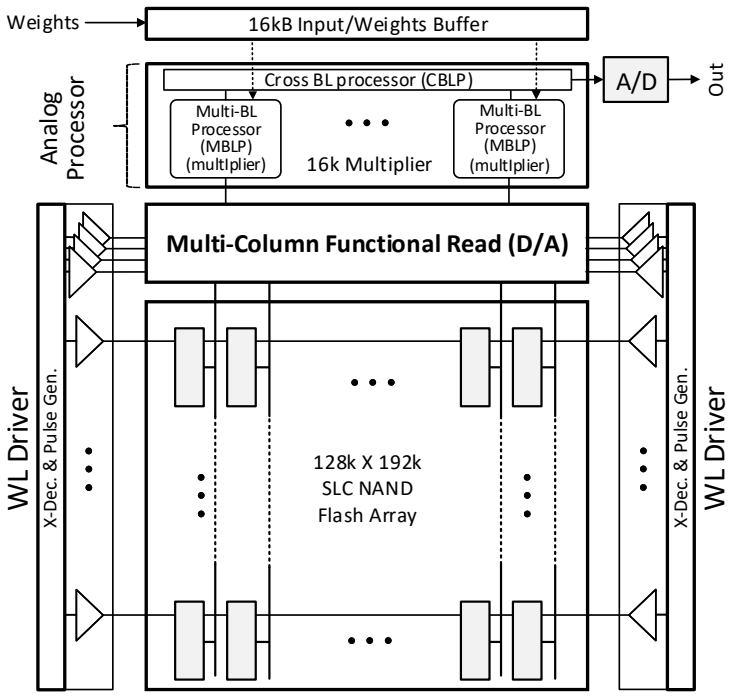


Figure 7.2: Proposed deep in-memory architecture for SLC NAND flash (DIMA-F).

voltage V_{read}, while other WLs in that block are set to voltage V_{pass}. In the precharge phase, the gate voltages of all SEL transistors are set to $V_{\text{pre}} + V_{\text{t}}$ and the gate voltage of PCH transistors is set to $V_{\text{dd}} + V_{\text{t}}$, charging the BLs and OUT nodes to approximately V_{pre} and V_{dd}, respectively. In the evaluation phase, PCH transistors are effectively turned off allowing the discharge of the C_{OUT} capacitor. The SEL transistors act as clamp transistors and are pulse width modulated such that the overall discharge on C_{OUT} is proportional to the decimal value of D. The current through string $I_{s,i} \approx I_{on}$ if $d_i = 1$ else $I_{s,i} \approx 0$. Choosing $T_i = 2^i T_0$ enables a D/A operation via MC-FR resulting in $\Delta V_{\text{OUT}} = \frac{I_{on} T_0}{C_{\text{OUT}}} \sum_{i=0}^{W-1} 2^i d_i$.

7.2.2 Multi-BL and Cross-BL Processing (MBLP and CBLP)

A capacitive multiplier similar to the one introduced in [27, 96] is used here (see Fig. 7.5). The analog value obtained from MC-FR is multiplied by a digital number stored in the input/weight buffer. The values from the input/weight buffer are given sequentially to the multiplier. The switching

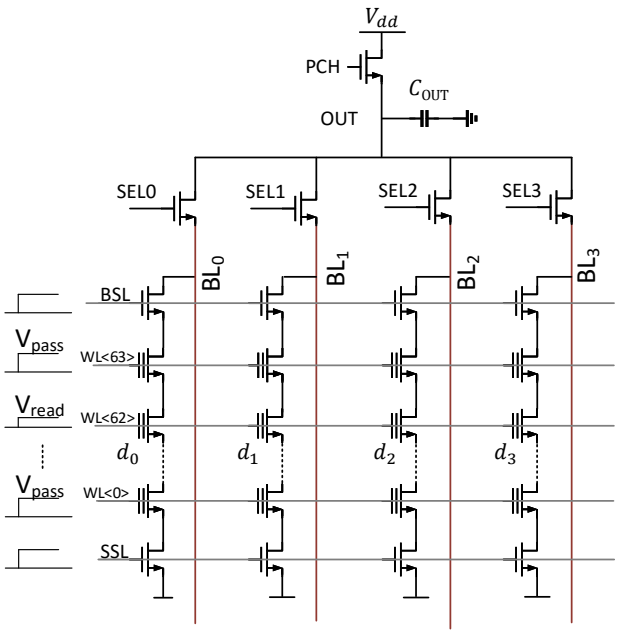


Figure 7.3: Architecture of the proposed MC-FR technique for $W = 4$.

sequence for multiplication is shown in Fig. 7.5. The multiplication process involves sequential charge sharing based on the digital inputs p_i. The effective voltage discharge of V_M is proportional to the product, $\Delta V_M = \sum_i (0.5)^i p_i \Delta V_{\mathrm{OUT}}$. Multiplier outputs across the plane are charge-shared to perform an average/addition operation on one of the two CBLP rails based on the sign of the outputs.

7.3 Evaluation Methodology

In this section, we present behavioral models, energy models, benchmarks and methodology employed for evaluation of DIMA-F.

7.3.1 Behavioral Models

Behavioral models are required to perform large-scale application-level simulations. These models need to account for non-idealities such as threshold voltage variations, read and program disturbance, diffusion, inter-cell interference (ICI), and back pattern dependency. We propose behavioral models that estimate effective string resistance by using long channel approxima-

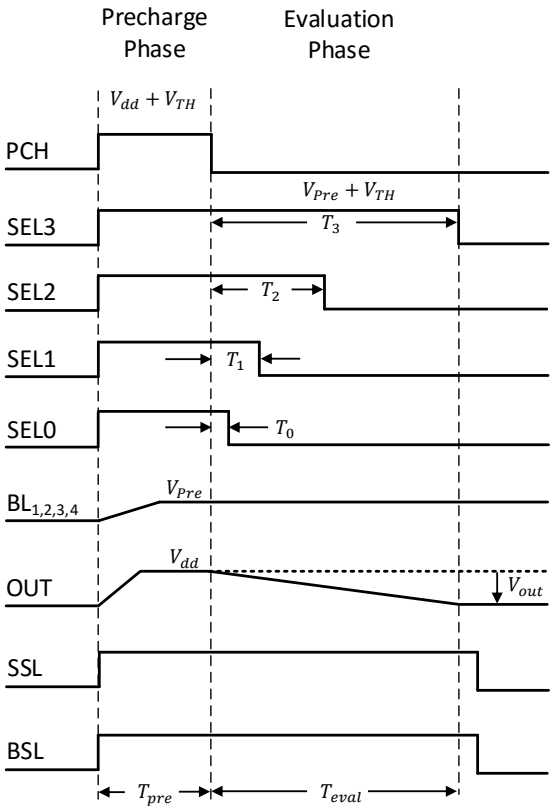


Figure 7.4: Timing of the proposed MC-FR technique for $W = 4$.

tion. These models are able to capture the behavior of NAND flash array at a string level which is sufficient for application-level simulations. Effective resistance of a cell during read $R(V_G)$ is estimated at super threshold by

$$R(V_G) \approx \big(k(V_G - V_{\rm t})\big)^{-1} \tag{7.1}$$

and at near threshold by

$$R(V_G) \approx \left(I_S\Big(\frac{W}{L}\Big)e^{\frac{V_G - V_{\rm t}}{nV_T}}\frac{V_{DS}}{V_T}\right)^{-1} \tag{7.2}$$

Equations (7.1) and (7.2) allow us to estimate the effective string current ($I_{\rm s}$) as a function of $V_{\rm read}$ using

$$I_{\rm s}(V_{\rm read}) = \frac{V_{\rm BL}}{R_k(V_{\rm read}) + \sum_{i=0, i\neq k}^{63} R_i(V_{\rm pass})} \tag{7.3}$$

where R_i is the effective resistance of the cell i, V_{BL} is the BL voltage, and k is the cell being read. While effects such as back-pattern dependencies are captured by the model, other variations are accounted by modeling the threshold voltage as a Gaussian random variable ($\mathbf{V}_{\mathrm{t}}$) [128]. Thus, cell resistance and the overall string current during the read operations are also treated as random variables, $\mathbf{R}_i$, and $\mathbf{I}_{\mathrm{s}}$. Hence, the output of the MC-FR is also a random variable:

$$\Delta \mathbf{V}_{\mathrm{OUT}}(V_{\mathrm{read}}) = \sum_{i=0}^{W-1} \frac{\mathbf{I}_{s,i}(V_{\mathrm{read}}) T_i}{C_{\mathrm{OUT}}} \tag{7.4}$$

V_{read} is chosen to minimize the mean squared error with respect to the ideal output.

7.3.2 Energy Models

We employ and build upon the energy models proposed in [129]. Energy consumption during MC-FR is dominated by the energy to charge BLs (E_{BL}), the energy to toggle WLs (E_{WL}) and the energy dissipation due to the string currents (E_{s}).

A conventional NAND flash suffers large stall times between page reads due to the limited speed of the shared I/O bus. The MC-FR technique reduces the read energy consumption as compared to conventional current sensing. Here, the lack of stall times prevent BLs from completely discharging between the consecutive read cycles which reduce E_{BL}. This reduces precharge times allowing throughput improvements, and reduction of E_{s}. Furthermore, the lack of stall times between consecutive reads within a block reduces the number of WL transitions between V_{read} and V_{pass} to two, as other WLs need not be discharged between reads. Thus, the average energy for a page read of DIMA-F $E_{\text{DIMA-F}}$ is

$$E_{\text{DIMA-F}} = E_{\mathrm{BL}} + E_{\mathrm{WL}} + E_{\mathrm{s}} \tag{7.5}$$

$$E_{\mathrm{BL}} = 0.5 N_{\mathrm{BL}} C_{\mathrm{BL}} \Delta V_{\mathrm{BL}} V_{\mathrm{dd}} \tag{7.6}$$

$$E_{\mathrm{WL}} = C_{\mathrm{WL}} V_{\mathrm{pass}} (V_{\mathrm{pass}} - V_{\mathrm{read}}) / \eta_{\mathrm{WL}} \tag{7.7}$$

$$E_{\mathrm{s}} = V_{\mathrm{dd}} I_{\mathrm{s,avg}} (T_{\mathrm{pre}} + T_{\mathrm{eval}}) \tag{7.8}$$

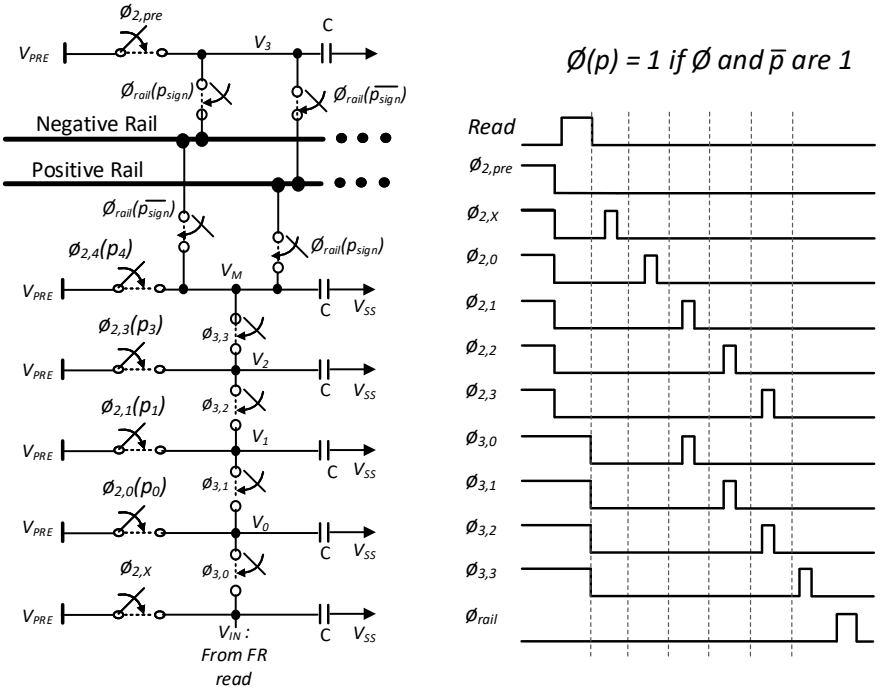


Figure 7.5: Signed multiplier and associated timing.

where $I_{\text{s,avg}}$ is the average string current, C_{BL} and C_{WL} are BL and WL capacitances, η_{WL} is the efficiency of the charge pump driving the WLs while T_{pre} and T_{eval} are the precharge and evaluation times, respectively.

7.3.3 Benchmarks

DIMA-F is tested over the following applications to demonstrate its benefits.

Linear support vector machine (SVM): Linear SVM is a simple and widely used classifier, which uses dot product for decisions. We use SVM for face detection task on the Caltech 101 database [126]. Linear SVM classification involves computing, $y = \boldsymbol{w}^T\boldsymbol{x} + b$, where $\boldsymbol{w}$ and b are pre-trained weights and $\boldsymbol{x}$ is the image vector to be classified. The image is classified as *face* if $y > 0$ and as *non-face* otherwise.

Cross-correlation based detection (CC): Cross-correlation is a useful metric to measure the similarity between two data vectors. We use CC as a distance metric in k-nearest neighbor (k-NN) algorithm for face recognition on Extended Yale B database [127]. It has 2336 test images with 28 classes.

7.3.4 Simulation Setup

For simulations in this chapter, NAND flash memory in a 32 nm node with 16 kB per pages, 64 pages per block, 3000 blocks per plane and 4 planes per IC is used. The images under test are scaled to 200×320, where each pixel is represented in 8-b fixed-point. Additionally, Extended Yale B images are pre-normalized for the CC based algorithms. Each image is rearranged into a 64 k pixel vector and stored on 4 pages across 4 planes. Input/weight buffer stores weights in the case of SVM and reference images in the case of CC. The dot product outputs are converted into digital domain via 8-b ADC for post-processing. Simulation methodology is described in Fig. 7.6. System-level simulations were performed using behavioral models described in Section 7.3.1 with model parameters obtained from SPICE simulations of a NAND flash array. Two architectures for a conventional baseline are considered to demonstrate the benefits of DIMA-F: (a) single NAND IC with an off-chip processor, and (b) a standard solid state drive (SSD) containing 16 ICs with an external processor.

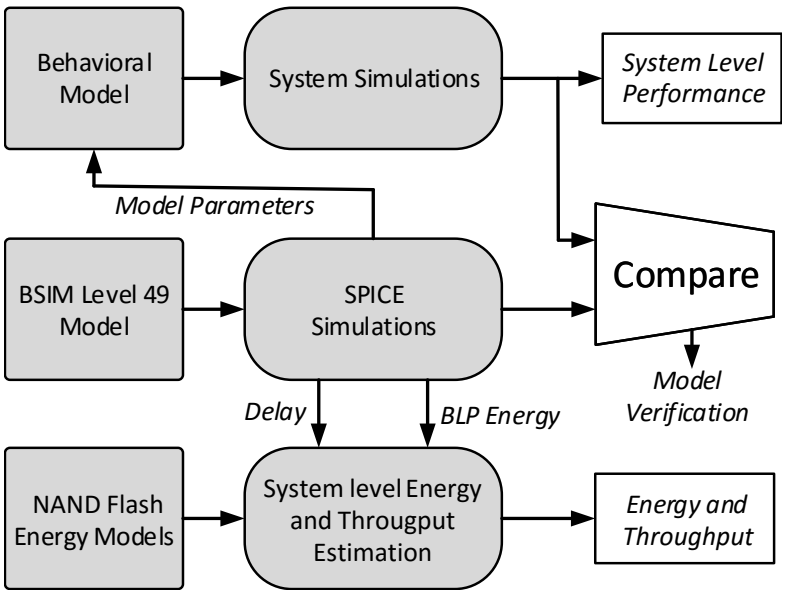


Figure 7.6: Simulation methodology.

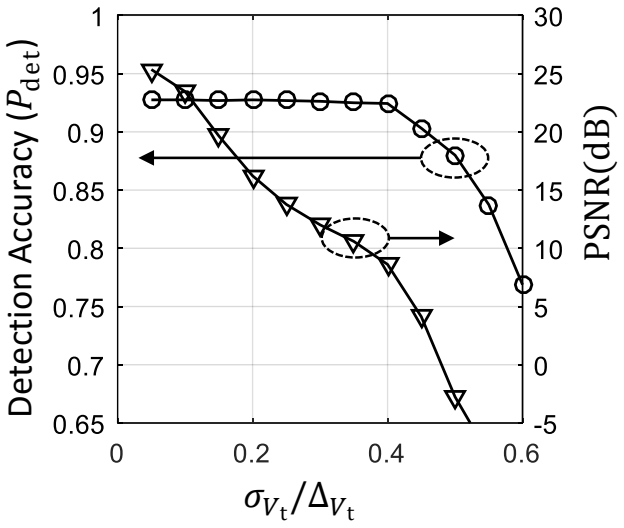


Figure 7.7: Detection accuracy P_{det} SVM algorithm as a function of threshold voltage variation.

7.4 Simulation Results

Figures 7.7 and 7.8 show the detection accuracy (P_{det}) of DIMA-F as a function of $\sigma_{V_t}/\Delta_{V_t}$, where σ_{V_t} is the variance of the threshold voltage and Δ_{V_t} is the mean threshold voltage difference between programmed and erased cells. Peak signal-to-noise ratio (PSNR) of the image read via MC-FR degrades with increasing threshold voltage variation. The SVM algorithm is more robust to threshold voltage variation. Threshold voltage variations ($\sigma_{V_t}/\Delta_{V_t}$) in a typical NAND flash memory ranges from 0.2 to 0.3. The detection accuracy of the SVM algorithm is 92% in this range.

Detection accuracy of the CC algorithm is about 88% for Top-1 and increases to 95% for Top-3 case in the $\sigma_{V_t}/\Delta_{V_t}$ range of 0.2 to 0.5, where Top-k is the accuracy when the correct label is among the top k candidates. The accuracy improved under the application of k-NN algorithm using CC as the distance metric. Detection accuracy of at least 92% for $k = 3$ and 95% for $k = 5$ is observed for $\sigma_{V_t}/\Delta_{V_t}$ in the range of 0.2 to 0.4.

7.4.1 Throughput and Energy

The throughput of individual conventional NAND IC is limited by the I/O that has a data transfer rate of 800MB/s (ONFI 4 standard [130]). In an

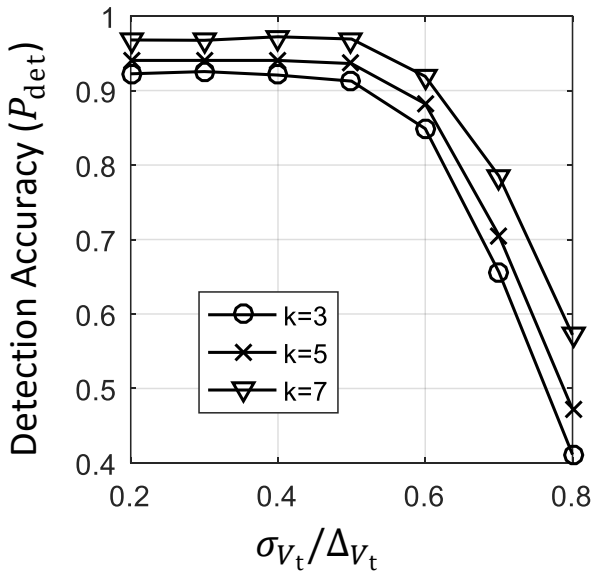


Figure 7.8: Detection accuracy P_{det} of CC based k-NN algorithm as a function of threshold voltage variation.

SSD the throughput is further limited by the PCIe bandwidth of 8GB/s [131]. Since DIMA-F based SSD does not have such I/O limitations, it can read at a rate of 7.56GB/s/IC. Therefore, a throughput improvement of 9× and 15× is achieved compared to a conventional system with a single IC and SSD scenario, respectively.

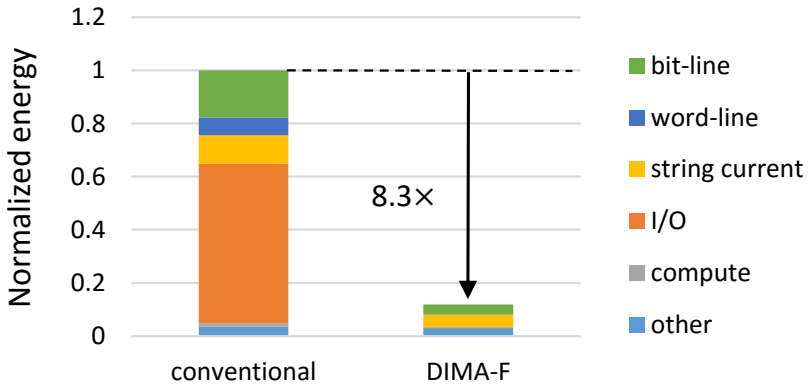


Figure 7.9: Estimated energy savings in the single IC scenario.

Energy estimates were obtained from models described in Section 7.3.2 with parameters obtained via SPICE simulations. I/O energy is estimated conservatively such that the device would meet the typical ONFI 4 standard. We observe that I/O energy is the dominating component of the conventional

system's energy consumption. In a single IC scenario, the I/O load is conservatively estimated to include the load of a single NAND IC connected to a bus. However, in an SSD scenario, multiple ICs are connected to the bus and would proportionally increase the I/O load. Overall 8.3× and 23× energy savings are achieved compared to the single IC scenario and SSD scenario, respectively. Energy breakdown for the single IC scenario is shown in Fig. 7.9.

7.5 Conclusions

Scaling trends in memory density and bandwidth suggest that the memory access problem will get worse over time. In-memory computing provides an alternative approach to address this problem. In this chapter we propose DIMA-F, which achieves 8×-to-23× energy savings and 9×-to-15× throughput improvements by overcoming the I/O barriers on SLC NAND flash memory. Future work includes extending DIMA-F to MLC NAND flash memories and other non-volatile memories.

CHAPTER 8

CONCLUSIONS AND FUTURE RESEARCH

Machine learning (ML) algorithms are increasingly prevalent in various applications, including embedded-sensor networks and multimedia applications such as object detection or speech recognition, to process massive data volumes. In particular, ML algorithms on Edge devices would open up numerous applications that would benefit society. This dissertation is a step towards enabling these ML algorithms in the Edge devices. One of the many challenges in doing so is the large computational complexity of ML algorithms, coupled with the resource constraints of these Edge devices.

This dissertation spans algorithmic, architectural, and circuit solutions to address this challenge. The key area of focus is mitigating the memory access energy and throughput bottlenecks that dominate the conventional digital architecture. In-memory architectures have emerged as an attractive platform to address memory access costs. This dissertation revolves around the energy-vs-accuracy trade-offs in in-memory architecture that stem from their use of mixed-signal computations.

8.1 Summary of Contributions

Our research adopts a cross-layered view of in-memory architectures, where energy consumption and the SNR of in-memory architecture can be traded off with each other using parameters that span across the compute stack from devices, circuits, architectures, and algorithms. The goal is to design in-memory architectures that meet the system-level accuracy requirements with minimum energy consumption and latency.

In order to design such a system, we first develop a methodology to reduce the precision requirements of DNNs. Then, the system-level trade-offs between energy and SNR of in-memory architectures are character-

ized. We also explore ways to enhance energy efficiency by: (a) reducing the application-level precision requirements or by (b) pushing the limits of energy-vs-accuracy trade-offs of in-memory architectures via compensation techniques.

First, an iterative mixed-precision quantization (IMPQ) methodology to reduce precision requirements of DNNs by allocating precision at kernel-wise granularity is developed. This problem is challenging due to the sheer number of possibilities. IMPQ involves iterative cycles of estimating the sensitivity of kernels, reducing the precision on the least sensitive kernels and retraining. The effectiveness of IMPQ was studied on compact networks such as ResNet-20 and MobileNet-V1, where state-of-the-art results were achieved both in terms of accuracy and effective reduction in precision.

Next, a compositional framework for in-memory architectures was proposed that identifies in-memory architectures as a combination of: (a) three *in-memory compute models* (charge accumulation (QA), current accumulation (QA) and charge sharing (QS)); and (b) four *mixed-signal arithmetic decomposition* methods (MM, BM, MB, BB) denoting the binarization of weights and/or activations and partitioning of a *multi-bit dot product* functionality into analog and digital computations.

This compositional framework is used to analyze and predict the SNR of the in-memory architectures as a function of various technology, architecture, and circuit parameters. This analysis enables us to make appropriate design choices to meet the system-level accuracy and energy requirements. Based on the analysis, a set of guidelines that designers could use in developing in-memory architectures was presented.

On-chip learning using stochastic gradient descent to adapt and track variations in PVT and data statistics is studied. Its benefits were demonstrated using a 65 nm CMOS IC prototype implementing the SVM-SGD algorithm. Measurements on the IC show that on-chip learned weights enable accurate inference in the presence of scaled BL voltage swing, thereby leading to a 2.4× greater energy savings over an off-chip trained DIMA implementation. Compared to a conventional fixed-function digital architecture with identical SRAM array, the prototype IC achieves up to 21× lower energy and 4.7× smaller delay leading to a 100× reduction in the EDP.

Single-Write In-memory Program-vErify (SWIPE) was proposed to address the impact of device variability in crossbar memories that stems from

spatial variations and cycle-to-cycle (C2C) variations (*write noise*). SWIPE achieves accurate bitcell writes for in-memory computing applications via a single scan of the array, thereby being 5×-to-10× more energy and latency efficient than program-verify methods.

Finally, deep in-memory architecture for NAND flash (DIMA-F) was proposed to extend in-memory computing to storage class memories. Simulations show that DIMA-F achieves between 8×-to-23× energy savings, 9×-to-15× throughput gain, and 72×-to-345× improvements in energy-delay product (EDP) compared to the conventional NAND flash architecture incorporating an external digital ASIC for computation.

8.2 Future Research Prospects

In-memory computing is a promising technology to enable artificial intelligence in the devices at the Edge. It is an emerging area of research that requires device, circuit, architecture, and algorithmic innovations. Few promising research directions include:

Rethinking Memory Hierarchy: Memory hierarchy in modern computers was designed to minimize the latency of memory access, i.e., to address the von Neumann bottleneck. Since in-memory architectures enable computations in a variety of memory technologies, numerous possibilities remain unexplored, and questions need to be answered in relation to the memory hierarchy. An exciting research direction is to find methods to introduce computing capability in each memory sub-system in the hierarchy. This may allow programmers to fetch a function of the data in addition to traditional load and store operations. If such a system is possible, one could rethink the memory hierarchy and its organization, as the computation is distributed across memory sub-systems.

Compute-Driven Memory Technologies: The computational framework presented in this dissertation allows us to predict the SNR and energy consumption of in-memory computation. The SNR of in-memory architectures is often limited by device variations. This is because memory designs primarily optimize storage density and for traditional memory operations. The compositional framework enables device researchers to identify aspects of their devices to improve upon in order to facilitate the design of in-memory

architectures. Furthermore, in-memory computing creates a unique space for device-architecture co-design that can result in orders-of-magnitude improvements in energy-efficiency and computational capabilities of computational platforms.

Architecture-Driven Algorithm Design: The computational framework reveals the conditions under which in-memory architectures are able to maximize their energy efficiency. Parameters such as bit-precision of weights and inputs, dot product lengths, and SNR requirements play an important role. Furthermore, data-reuse opportunities and DNN architectures also have implications on the energy-efficiency of in-memory architectures. A potential extension would be to expand the methodology in Chapter 2 to search for the best DNN for in-memory architectures. AutoML techniques such as Neural Architecture Search [132] can also be employed for this purpose.

Applications beyond Machine Learning: Machine learning is an attractive workload for in-memory architectures due to their ability to tolerate noise and non-idealities. However, memory access challenges exist in numerous other workloads beyond machine learning, such as those in statistical signal processing. Extending in-memory architectures to such workloads comes with its challenges. Examples of such applications include synthetic aperture radar (SAR) processing [133], simultaneous localization and mapping (SLAM) [134], and communication systems. Key challenges here are: (a) higher precision requirements, and (b) availability of efficient signal processing algorithms for digital architectures.

REFERENCES

[1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot et al., "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, p. 484, 2016.

[2] N. Izrael, "Prognosis for health care IoT: Six predictions for 2019," *Forbes*, 2019. [Online]. Available: https://www.forbes.com/sites/forbestechcouncil/2019/03/01/prognosis-for-health-care-iot-six-predictions-for-2019/#792af27fdddb

[3] C. Wayne Crews Jr, "Helicopter government? How the internet of things enables pushbutton regulation from a distance," *Forbes*, 2019. [Online]. Available: https://www.forbes.com/sites/waynecrews/2019/11/11/helicopter-government/#96559ddf6673

[4] N. Joshi, "How IoT and AI can enable environmental sustainability," *Forbes*, 2019. [Online]. Available: https://www.forbes.com/sites/cognitiveworld/2019/09/04/how-iot-and-ai-can-enable-environmental-sustainability/#38d0128e68df

[5] S. Chandler, "How the internet of things will help fight climate change," *Forbes*, 2019. [Online]. Available: https://www.forbes.com/sites/simonchandler/2019/11/05/how-the-internet-of-things-will-help-fight-climate-change/#4126702858a3

[6] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *IEEE International Solid-State Circuits Conference (ISSCC)*, 2014, pp. 10–14.

[7] B. Gu, A. S. Yoon, D.-H. Bae, I. Jo, J. Lee, J. Yoon, J.-U. Kang, M. Kwon, C. Yoon, S. Cho et al., "Biscuit: A framework for near-data processing of big data workloads," in *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3. IEEE Press, 2016, pp. 153–165.

[8] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan 2017.

[9] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, "ENVISION: A 0.26-to-10 TOPS/W subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28nm FDSOI," in *IEEE International Solid-State Circuits Conference (ISSCC)*, 2017, pp. 246–247.

[10] D. Shin, J. Lee, J. Lee, and H.-J. Yoo, "DNPU: An 8.1 TOPS/W reconfigurable CNN-RNN processor for general-purpose deep neural networks," in *IEEE International Solid-State Circuits Conference (ISSCC)*, 2017, pp. 240–241.

[11] P. N. Whatmough, S. K. Lee, H. Lee, S. Rama, D. Brooks, and G.-Y. Wei, "A 28nm SOC with a 1.2 GHz 568nJ/prediction sparse deep-neural-network engine with> 0.1 timing error rate tolerance for IOT applications," in *IEEE International Solid-State Circuits Conference (ISSCC)*, 2017, pp. 242–243.

[12] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo, "UNPU: A 50.6 TOPS/W unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision," in *IEEE International Solid-State Circuits Conference (ISSCC)*, 2018, pp. 218–220.

[13] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," *International conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, vol. 49, no. 4, pp. 269–284, 2014.

[14] Y. Zhang, L. Xu, Q. Dong, J. Wang, D. Blaauw, and D. Sylvester, "Recryptor: A reconfigurable cryptographic cortex-m0 processor with in-memory and near-memory computing for iot security," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 4, pp. 995–1005, 2018.

[15] K. C. Akyel, H.-P. Charles, J. Mottin, B. Giraud, G. Suraci, S. Thuries, and J.-P. Noel, "Drc 2: Dynamically reconfigurable computing circuit based on memory architecture," in *2016 IEEE International Conference on Rebooting Computing (ICRC)*. IEEE, 2016, pp. 1–8.

[16] J. Wang, X. Wang, C. Eckert, A. Subramaniyan, R. Das, D. Blaauw, and D. Sylvester, "A compute SRAM with bit-serial integer/floating-point operations for programmable in-memory vector acceleration," in *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2019, pp. 224–226.

[17] M. M. Shulaker, T. F. Wu, A. Pal, L. Zhao, Y. Nishi, K. Saraswat, H.-S. P. Wong, and S. Mitra, "Monolithic 3D integration of logic and memory: Carbon nanotube FETs, resistive RAM, and silicon FETs," in *2014 IEEE International Electron Devices Meeting*. IEEE, 2014, pp. 27–4.

[18] J. Jeddeloh and B. Keeth, "Hybrid memory cube new dram architecture increases density and performance," in *IEEE Symposium on VLSI Technology (VLSI Technology)*, 2012.

[19] S. Jeloka, N. B. Akesh, D. Sylvester, and D. Blaauw, "A 28 nm configurable memory (TCAM/BCAM/SRAM) using push-rule 6T bit cell enabling logic-in-memory," *IEEE Journal of Solid-State Circuits*, vol. 51, no. 4, pp. 1009–1021, 2016.

[20] I. Hayashi, T. Amano, N. Watanabe, Y. Yano, Y. Kuroda, M. Shirata, K. Dosaka, K. Nii, H. Noda, and H. Kawai, "A 250-MHz 18-Mb full ternary CAM with low-voltage matchline sensing scheme in 65-nm cmos," *IEEE journal of solid-state circuits*, vol. 48, no. 11, pp. 2671–2680, 2013.

[21] K. Nii, T. Amano, N. Watanabe, M. Yamawaki, K. Yoshinaga, M. Wada, and I. Hayashi, "A 28nm 400MHz 4-parallel 1.6 Gsearch/s 80Mb ternary CAM," in *IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. IEEE, 2014, pp. 240–241.

[22] J. Li, R. Montoye, M. Ishii, K. Stawiasz, T. Nishida, K. Maloney, G. Ditlow, S. Lewis, T. Maffitt, R. Jordan et al., "1Mb 0.41 μm 2 2T-2R cell nonvolatile TCAM with two-bit encoding and clocked self-referenced sensing," in *Symposium on VLSI Technology (VLSI-T)*. IEEE, 2013, pp. C104–C105.

[23] S. Matsunaga, S. Miura, H. Honjou, K. Kinoshita, S. Ikeda, T. Endoh, H. Ohno, and T. Hanyu, "A 3.14 um 2 4T-2MTJ-cell fully parallel TCAM based on nonvolatile logic-in-memory architecture," in *Symposium on VLSI Circuits (VLSI-C)*. IEEE, 2012, pp. 44–45.

[24] L.-Y. Huang, M.-F. Chang, C.-H. Chuang, C.-C. Kuo, C.-F. Chen, G.-H. Yang, H.-J. Tsai, T.-F. Chen, S.-S. Sheu, K.-L. Su et al., "ReRAM-based 4T2R nonvolatile TCAM with 7x NVM-stress reduction, and 4x improvement in speed-wordlength-capacity for normally-off instant-on filter-based search engines used in big-data processing," in *Symposium on VLSI Circuits (VLSI-C)*. IEEE, 2014, pp. 1–2.

[25] M. Natsui, D. Suzuki, N. Sakimura, R. Nebashi, Y. Tsuji, A. Morioka, T. Sugibayashi, S. Miura, H. Honjo, K. Kinoshita et al., "Nonvolatile logic-in-memory array processor in 90nm MTJ/MOS achieving 75% leakage reduction using cycle-based power gating," in *IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2013, pp. 194–195.

[26] M. Kang, M.-S. Keel, N. R. Shanbhag, S. Eilert, and K. Curewitz, "An energy-efficient VLSI architecture for pattern recognition via deep embedding of computation in SRAM," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2014, pp. 8326–8330.

[27] M. Kang, S. K. Gonugondla, M.-S. Keel, and N. R. Shanbhag, "An energy-efficient memory-based high-throughput VLSI architecture for Convolutional Networks," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2015.

[28] M. Kang, S. K. Gonugondla, A. Patil, and N. R. Shanbhag, "A multi-functional in-memory inference processor using a standard 6T SRAM array," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 2, pp. 642–655, Feb 2018.

[29] A. Biswas and A. P. Chandrakasan, "Conv-RAM: An energy-efficient SRAM with embedded convolution computation for low-power CNN-based machine learning applications," in *IEEE International Solid-State Circuits Conference (ISSCC)*, 2018, pp. 488–490.

[30] J. Zhang, Z. Wang, and N. Verma, "In-memory computation of a machine-learning classifier in a standard 6T SRAM array," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 4, pp. 915–924, April 2017.

[31] H. Valavi, P. J. Ramadge, E. Nestler, and N. Verma, "A mixed-signal binarized convolutional-neural-network accelerator integrating dense weight storage and multiplication for reduced data movement," in *IEEE Symposium on VLSI Circuits (VLSI-C)*. IEEE, 2018, pp. 141–142.

[32] W.-S. Khwa, J.-J. Chen, J.-F. Li, X. Si, E.-Y. Yang, X. Sun, R. Liu, P.-Y. Chen, Q. Li, S. Yu et al., "A 65nm 4kb algorithm-dependent computing-in-memory sram unit-macro with 2.3 ns and 55.8 tops/w fully parallel product-sum operation for binary dnn edge processors," in *IEEE International Solid-State Circuits Conference (ISSCC)*, 2018, pp. 496–498.

[33] Z. Jiang, S. Yin, M. Seok, and J.-S. Seo, "XNOR-SRAM: In-memory computing SRAM macro for binary/ternary deep neural networks," in *IEEE Symposium on VLSI Technology (VLSI-C)*. IEEE, 2018, pp. 173–174.

[34] X. Si, J.-J. Chen, Y.-N. Tu, W.-H. Huang, J.-H. Wang, Y.-C. Chiu, W.-C. Wei, S.-Y. Wu, X. Sun, R. Liu et al., "A twin-8T SRAM computation-in-memory macro for multiple-bit CNN-based machine learning," in *IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2019, pp. 396–398.

[35] H. Jia, Y. Tang, H. Valavi, J. Zhang, and N. Verma, "A microprocessor implemented in 65nm CMOS with configurable and bit-scalable accelerator for programmable in-memory computing," *arXiv preprint arXiv:1811.04047*, 2018.

[36] S. Okumura, M. Yabuuchi, K. Hijioka, and K. Nose, "A ternary based bit scalable, 8.80 TOPS/W CNN accelerator with many-core processing-in-memory architecture with 896k synapses/mm2," in *2019 IEEE Symposium on VLSI Circuits*. IEEE, 2019, pp. 248–249.

[37] J. Kim, J. Koo, T. Kim, Y. Kim, H. Kim, S. Yoo, and J.-J. Kim, "Area-efficient and variation-tolerant in-memory BNN computing using 6T SRAM array," in *IEEE Symposium on VLSI Circuits (VLSI-C)*. IEEE, 2019, pp. 118–119.

[38] R. Guo, Y. Liu, S. Zheng, S.-Y. Wu, P. Ouyang, W.-S. Khwa, X. Chen, J.-J. Chen, X. Li, L. Liu, M.-F. Chang, S. Wei, and S. Yin, "A 5.1pJ/neuron 127.3μs/inference RNN-based speech recognition processor using 16 computing-in-memory SRAM macros in 65nm CMOS," in *2019 IEEE Symposium on VLSI Circuits*. IEEE, 2019, pp. 120–121.

[39] J. Yue, Z. Yuan, X. Feng, Y. He, Z. Zhang, X. Si, R. Liu, M.-F. Chang, X. Li, H. Yang, and Y. Liu, "A 65nm computing-in-memory-based CNN processor with 2.9-to-35.8TOPS/W system energy efficiency using dynamic-sparsity performance-scaling architecture and energy-efficient inter/intra-macro data reuse," in *IEEE International Solid-State Circuits Conference (ISSCC)*, 2020, pp. 234–235.

[40] J.-W. Su, X. Si, Y.-C. Chou, T.-W. Chang, W.-H. Huang, Y.-N. Tu, R. Liu, T.-W. Lu, Pei-Jungand Liu, J.-H. Wang, Z. Zhang, H. Jiang, S. Huang, C.-C. Lo, R.-S. Liu, C.-C. Hsieh, K.-T. Tang, S.-S. Sheu, S.-H. Li, H.-Y. Lee, S.-C. Chang, S. Yu, and M.-F. Chang, "A 28nm 64Kb inference-training two-way transpose multibit 6T SRAM compute-in-memory macro for AI edge chips," in *IEEE International Solid-State Circuits Conference (ISSCC)*, 2020, pp. 240–241.

[41] Q. Dong, M. E. Sinangil, B. Erbagci, D. Sun, W.-S. Khwa, H.-J. Liao, Y. Wang, and J. Chang, "A 351 TOPS/W and 372.4 GOPS compute-in-memory sram macro in 7nm FinFET CMOS for machine learning applications," in *IEEE International Solid-State Circuits Conference (ISSCC)*, 2020, pp. 242–243.

[42] X. Si, Y.-N. Tu, W.-H. Huang, J.-W. Su, P.-J. Lu, J.-H. Wang, T.-W. Liu, S.-Y. Wu, R. Liu, Y.-C. Chou, Z. Zhang, S.-H. Sie, W.-C. Wei, Y.-C. Lo, T.-H. Wen, T.-H. Hsu, Y.-K. Chen, W. Shih, C.-C. Lo, R.-S. Liu, C.-C. Hsieh, K.-T. Tang, N.-C. Lien, W.-C. Shih, Y. He, Q. Li, and M.-F. Chang, "A 28nm 64Kb 6T SRAM computing-in- memory macro with 8b MAC operation for AI edge chips," in *IEEE International Solid-State Circuits Conference (ISSCC)*, 2020, pp. 246–247.

[43] W.-H. Chen, K.-X. Li, W.-Y. Lin, K.-H. Hsu, P.-Y. Li, C.-H. Yang, C.-X. Xue, E.-Y. Yang, Y.-K. Chen, Y.-S. Chang et al., "A 65nm 1Mb nonvolatile computing-in-memory ReRAM macro with sub-16ns multiply-and-accumulate for binary DNN AI edge processors," in *IEEE International Solid-State Circuits Conference (ISSCC)*, 2018, pp. 494–496.

[44] L. Fick, D. Blaauw, D. Sylvester, S. Skrzyniarz, M. Parikh, and D. Fick, "Analog in-memory subthreshold deep neural network accelerator," in *2017 IEEE Custom Integrated Circuits Conference (CICC)*. IEEE, 2017, pp. 1–4.

[45] C.-X. Xue, W.-H. Chen, J.-S. Liu, J.-F. Li, W.-Y. Lin, W.-E. Lin, J.-H. Wang, W.-C. Wei, T.-W. Chang, T.-C. Chang et al., "A 1Mb multibit ReRAM computing-in-memory macro with 14.6 ns parallel MAC computing time for CNN based AI edge processors," in *IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2019, pp. 388–390.

[46] B. Yan, Q. Yang, W.-H. Chen, K.-T. Chang, J.-W. Su, C.-H. Hsu, S.-H. Li, H.-Y. Lee, S.-S. Sheu, M.-S. Ho et al., "RRAM-based spiking nonvolatile computing-in-memory processing engine with precision-configurable in situ nonlinear activation," in *2019 Symposium on VLSI Technology*. IEEE, 2019, pp. T86–T87.

[47] Y. Zha, E. Nowak, and J. Li, "Liquid Silicon: a nonvolatile fully programmable processing-in-memory processor with monolithically integrated ReRAM for big data/machine learning applications," in *IEEE Symposium on VLSI Circuits (VLSI-C)*. IEEE, 2019, pp. 206–207.

[48] C.-X. Xue, T.-Y. Huang, J.-S. Liu, T.-W. Chang, H.-Y. Kao, J.-H. Wang, T.-W. Liu, S.-Y. Wei, S.-P. Huang, W.-C. Wei, Y.-R. Chen, T.-H. Hsu, Y.-K. Chen, Y.-C. Lo, T.-H. Wen, C.-C. Lo, R.-S. Liu, C.-C. Hsieh, K.-T. Tang, and M.-F. Chang, "A 22nm 2Mb ReRAM compute-in-memory macro with 121-28TOPS/W for multibit MAC computing for tiny AI edge devices," in *IEEE International Solid-State Circuits Conference (ISSCC)*, 2020, pp. 244–245.

[49] S. K. Gonugondla, M. Kang, and N. R. Shanbhag, "A variation-tolerant in-memory machine learning classifier via on-chip training," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 11, pp. 3163–3173, 2018.

[50] M. Kang, S. Gonugondla, and N. R. Shanbhag, *Deep In-memory Architectures for Machine Learning*. Springer, 2020.

[51] M. Kang, S. Lim, S. Gonugondla, and N. R. Shanbhag, "An in-memory VLSI architecture for convolutional neural networks," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 3, pp. 494–505, 2018.

[52] M. Kang, S. K. Gonugondla, and N. R. Shanbhag, "A 19.4 nJ/decision 364K decisions/s in-memory random forest classifier in 6T SRAM array," in *IEEE European Solid-State Circuits Conference (ESSCIRC)*, 2017, pp. 263–266.

[53] S. K. Gonugondla, M. Kang, and N. Shanbhag, "A 42pJ/decision 3.12 TOPS/W robust in-memory machine learning classifier with on-chip training," in *IEEE International Solid-State Circuits Conference (ISSCC)*, 2018, pp. 490–492.

[54] P. Srivastava, M. Kang, S. K. Gonugondla, S. Lim, J. Choi, V. Adve, N. S. Kim, and N. Shanbhag, "PROMISE: An end-to-end design of a programmable mixed-signal accelerator for machine-learning algorithms," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*, 2018, pp. 43–56.

[55] M. Kang, S. K. Gonugondla, S. Lim, and N. R. Shanbhag, "A 19.4-nJ/decision, 364-K decisions/s, in-memory random forest multi-class inference accelerator," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 7, pp. 2126–2135, July 2018.

[56] H. Dbouk, S. Gonugondla, C. Sakr, and N. Shanbhag, "Keyram: A 0.34 uj/decision 18 k decisions/s recurrent attention in-memory processor for keyword spotting," in *IEEE Custom Integrated Circuits Conference (CICC)*, 2020, pp. 10–14.

[57] S. K. Gonugondla, B. Shim, and N. R. Shanbhag, "Perfect error compensation via algorithmic error cancellation," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2016, pp. 966–970.

[58] N. R. Shanbhag, N. Verma, Y. Kim, A. D. Patil, and L. R. Varshney, "Shannon-inspired statistical computing for the nanoscale era," *Proceedings of the IEEE*, vol. 107, no. 1, pp. 90–107, 2018.

[59] R. Sarpeshkar, "Analog versus digital: extrapolating from electronics to neurobiology," *Neural Computation*, vol. 10, no. 7, pp. 1601–1638, 1998.

[60] N. C. Wang, S. K. Gonugondla, I. Nahlus, N. R. Shanbhag, and E. Pop, "GDOT: A graphene-based nanofunction for dot-product computation," in *2016 IEEE Symposium on VLSI Technology*. IEEE, 2016, pp. 1–2.

[61] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2015, pp. 1135–1143.

[62] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[63] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and $<$ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[64] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 6848–6856.

[65] G. Huang, S. Liu, L. Van der Maaten, and K. Q. Weinberger, "Condensenet: An efficient densenet using learned group convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 2752–2761.

[66] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," *arXiv preprint arXiv:1605.04711*, 2016.

[67] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," *arXiv preprint arXiv:1612.01064*, 2016.

[68] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "DoReFa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.

[69] J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan, "PACT: Parameterized clipping activation for quantized neural networks," *arXiv preprint arXiv:1805.06085*, 2018.

[70] C. Sakr and N. Shanbhag, "An analytical method to determine minimum per-layer precision of deep neural networks," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018, pp. 1090–1094.

[71] Q. Lou, F. Guo, L. Liu, M. Kim, and L. Jiang, "AutoQ: Automated kernel-wise neural network quantization," *arXiv preprint arXiv:1902.05690*, 2019.

[72] Z. Dong, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer, "HAWQ: Hessian aware quantization of neural networks with mixed-precision," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 293–302.

[73] C. Sakr, Y. Kim, and N. Shanbhag, "Analytical guarantees on numerical precision of deep neural networks," in *International Conference on Machine Learning*, 2017, pp. 3007–3016.

[74] D. Lin, S. Talathi, and S. Annapureddy, "Fixed point quantization of deep convolutional networks," in *International Conference on Machine Learning*, 2016, pp. 2849–2858.

[75] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2016, pp. 4107–4115.

[76] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-net: Imagenet classification using binary convolutional neural networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer, 2016, pp. 525–542.

[77] A. Zhou, A. Yao, K. Wang, and Y. Chen, "Explicit loss-error-aware quantization for low-bit deep neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 9426–9435.

[78] D. Zhang, J. Yang, D. Ye, and G. Hua, "LQ-nets: Learned quantization for highly accurate and compact deep neural networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 365–382.

[79] X. Lin, C. Zhao, and W. Pan, "Towards accurate binary convolutional neural network," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017, pp. 345–353.

[80] B. Wu, Y. Wang, P. Zhang, Y. Tian, P. Vajda, and K. Keutzer, "Mixed precision quantization of convnets via differentiable neural architecture search," *arXiv preprint arXiv:1812.00090*, 2018.

[81] Y. Zhou, S.-M. Moosavi-Dezfooli, N.-M. Cheung, and P. Frossard, "Adaptive quantization for deep neural network," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[82] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "HAQ: Hardware-aware automated quantization with mixed precision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 8612–8620.

[83] A. Krizhevsky, G. Hinton et al., "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.

[84] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein et al., "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[85] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[86] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[87] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training deep neural networks with binary weights during propagations," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2015, pp. 3123–3131.

[88] C. Baskin, E. Schwartz, E. Zheltonozhskii, N. Liss, R. Giryes, A. M. Bronstein, and A. Mendelson, "Uniq: Uniform noise injection for non-uniform quantization of neural networks," *arXiv preprint arXiv:1804.10969*, 2018.

[89] C. Louizos, M. Reisser, T. Blankevoort, E. Gavves, and M. Welling, "Relaxed quantization for discretized neural networks," *arXiv preprint arXiv:1810.01875*, 2018.

[90] J. Woo and S. Yu, "Comparative study of cross-point MRAM array with exponential and threshold selectors for read operation," *IEEE Electron Device Letters*, vol. 39, no. 5, pp. 680–683, 2018.

[91] W. Wu, H. Wu, B. Gao, P. Yao, X. Zhang, X. Peng, S. Yu, and H. Qian, "A methodology to improve linearity of analog RRAM for neuromorphic computing," in *2018 IEEE Symposium on VLSI Technology*. IEEE, 2018, pp. T103–T104.

[92] A. D. Patil, H. Hua, S. Gonugondla, M. Kang, and N. R. Shanbhag, "An MRAM-based deep in-memory architecture for deep neural networks," in *International Symposium on Circuits and Systems (ISCAS)*, 2019, pp. 1–5.

[93] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *International Symposium on Computer Architecture (ISCA)*, 2016, pp. 14–26.

[94] V. Tripathi and B. Murmann, "Mismatch characterization of small metal fringe capacitors," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 8, pp. 2236–2242, 2014.

[95] G. Wegmann, E. A. Vittoz, and F. Rahali, "Charge injection in analog MOS switches," *IEEE Journal of Solid-State Circuits*, vol. 22, no. 6, pp. 1091–1097, 1987.

[96] M. Kang, S. Gonugondla, A. Patil, and N. Shanbhag, "A 481 pJ/decision 3.4M decision/s multifunctional deep in-memory inference processor using standard 6T SRAM array," *arXiv preprint arXiv:1610.07501*, 2016.

[97] ITRS-collaborations, "ITRS roadmap tables," *ITRS*, 2015. [Online]. Available: http://www.itrs2.net/itrs-reports.html

[98] H. Fujiwara, C.-Y. Lin, H.-Y. Pan, C.-H. Lin, P.-Y. Huang, K.-C. Lin, J.-J. Liaw, Y.-H. Chen, H.-J. Liao, and J. Chang, "A 7nm 2.1 GHz dual-port SRAM with WL-RC optimization and dummy-read-recovery circuitry to mitigate read-disturb-write issue," in *IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2019, pp. 390–392.

[99] B. Murmann, "Trends in low-power, digitally assisted A/D conversion," *IEICE Transactions on Electronics*, vol. 93, no. 6, pp. 718–729, 2010.

[100] B. Murmann, "A/D converter trends: Power dissipation, scaling and digitally assisted architectures," in *2008 IEEE Custom Integrated Circuits Conference*. IEEE, 2008, pp. 105–112.

[101] R. Schreier, G. C. Temes et al., *Understanding Delta-Sigma Data Converters*. IEEE press Piscataway, NJ, 2005, vol. 74.

[102] B. Murmann, "The race for the extra decibel: a brief review of current ADC performance trajectories," *IEEE Solid-State Circuits Magazine*, vol. 7, no. 3, pp. 58–66, 2015.

[103] B. Murmann, "ADC performance survey 1997-2019," 2019. [Online]. Available: https://web.stanford.edu/~murmann/adcsurvey.html

[104] D. Bankman and B. Murmann, "An 8-bit, 16 input, 3.2 pJ/op switched-capacitor dot product circuit in 28-nm FDSOI CMOS," in *Solid-State Circuits Conference (A-SSCC), 2016 IEEE Asian*. IEEE, 2016, pp. 21–24.

[105] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *arXiv preprint arXiv:1606.04838*, 2016.

[106] C. Radhakrishnan and S. Gonugondla, "Adaptive filtering in in-memory-based architectures," in *Asilomar conference on Signals, Systems and Computer*, 2019.

[107] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[108] C. Sakr, A. Patil, S. Zhang, Y. Kim, and N. Shanbhag, "Minimum precision requirements for the SVM-SGD learning algorithm," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017, pp. 1138–1142.

[109] "Center for biological and computational learning (CBCL) at MIT," 2000, `http://cbcl.mit.edu/software-datasets/index.html`.

[110] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner et al., "Razor: A low-power pipeline based on circuit-level timing speculation," in *IEEE/ACM International Symposium on Microarchitecture*, 2003, p. 7.

[111] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," in *International Symposium on Computer Architecture (ISCA)*, 2016, pp. 27–39.

[112] A. Ankit, I. E. Hajj, S. R. Chalamalasetti, G. Ndu, M. Foltin, R. S. Williams, P. Faraboschi, W.-m. W. Hwu, J. P. Strachan, K. Roy et al., "PUMA: A programmable ultra-efficient memristor-based accelerator for machine learning inference," in *International conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2019, pp. 715–731.

[113] L. Gao, P.-Y. Chen, and S. Yu, "Programming protocol optimization for analog weight tuning in resistive memories," *IEEE Electron Device Letters*, vol. 36, no. 11, pp. 1157–1159, 2015.

[114] Z. He, J. Lin, R. Ewetz, J.-S. Yuan, and D. Fan, "Noise injection adaption: End-to-end reram crossbar non-ideal effect adaption for neural network mapping," in *Design and Automation Conference (DAC)*, 2019, p. 57.

[115] Z. Zhu, H. Sun, Y. Lin, G. Dai, L. Xia, S. Han, Y. Wang, and H. Yang, "A configurable multi-precision CNN computing framework based on single bit RRAM," in *Design and Automation Conference (DAC)*, 2019, p. 56.

[116] Z. Jiang, Y. Wu, S. Yu, L. Yang, K. Song, Z. Karim, and H.-S. P. Wong, "A compact model for metal–oxide resistive random access memory with experiment verification," *IEEE Transactions on Electron Devices*, vol. 63, no. 5, pp. 1884–1892, 2016.

[117] M. Cheng, L. Xia, Z. Zhu, Y. Cai, Y. Xie, Y. Wang, and H. Yang, "TIME: A training-in-memory architecture for RRAM-based deep neural networks," *IEEE Transactionss on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 5, pp. 834–847, 2018.

[118] S. Yu, X. Guan, and H.-S. P. Wong, "On the switching parameter variation of metal oxide rram—part ii: Model corroboration and device design strategy," *IEEE Transactions on Electron Devices*, vol. 59, no. 4, pp. 1183–1188, 2012.

[119] B. Zhang, L.-Y. Chen, and N. Verma, "Stochastic data-driven hardware resilience to efficiently train inference models for stochastic hardware implementations," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 1388–1392.

[120] Y. Long, X. She, and S. Mukhopadhyay, "Design of reliable DNN accelerator with un-reliable ReRAM," in *Design, Automation and Test in Europe Conference (DATE)*, 2019, pp. 1769–1774.

[121] X. Sun and S. Yu, "Impact of non-ideal characteristics of resistive synaptic devices on implementing convolutional neural networks," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems (JETCAS)*, vol. 9, no. 3, pp. 570–579, 2019.

[122] F. Alibart, L. Gao, B. D. Hoskins, and D. B. Strukov, "High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm," *Nanotechnology*, vol. 23, no. 7, p. 075201, 2012.

[123] M. Hu, C. E. Graves, C. Li, Y. Li, N. Ge, E. Montgomery, N. Davila, H. Jiang, R. S. Williams, J. J. Yang et al., "Memristor-based analog computation and neural network classification with a dot product engine," *Advanced Materials*, vol. 30, no. 9, p. 1705914, 2018.

[124] L. Song, X. Qian, H. Li, and Y. Chen, "Pipelayer: A pipelined reram-based accelerator for deep learning," in *International Symposium on High-Performance Computer Architecture (HPCA)*, 2017, pp. 541–552.

[125] S. K. Gonugondla, M. Kang, Y. Kim, M. Helm, S. Eilert, and N. Shanbhag, "Energy-efficient deep in-memory architecture for NAND flash memories," in *International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2018, pp. 1–5.

[126] R. F. L. Fei-Fei and P. Perona, "One-shot learning of object categories," *IEEE Trans. Pattern Recognition and Machine Intelligence.*, 2004.

[127] A. Georghiades, P. Belhumeur, and D. Kriegman, "From few to many: Illumination cone models for face recognition under variable lighting and pose," *IEEE Trans. Pattern Anal. Mach. Intelligence*, vol. 23, no. 6, pp. 643–660, 2001.

[128] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai, "Threshold voltage distribution in MLC NAND flash memory: Characterization, analysis, and modeling," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013*. IEEE, 2013, pp. 1285–1290.

[129] V. Mohan, T. Bunker, L. Grupp, S. Gurumurthi, M. R. Stan, and S. Swanson, "Modeling power consumption of NAND Flash memories using Flashpower," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 7, pp. 1031–1044, 2013.

[130] ONFI-Workgroup, "Open nand flash interface specification," *ONFI Standard*, 2014. [Online]. Available: http://www.onfi.org/-/media/client/onfi/specs/onfi_4_2-gold.pdf?la=en&rev=1fb1f47489584c0eb2f31f22a30b94f8

[131] E. Doller, A. Akel, J. Wang, K. Curewitz, and S. Eilert, "DataCenter 2020: Near-memory acceleration for data-oriented applications," in *IEEE Symposium on VLSI Circuits Digest of Technical Papers*, 2014, pp. 1–4.

[132] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[133] M. Soumekh, *Synthetic Aperture Radar Signal Processing*. New York: Wiley, 1999, vol. 7.

[134] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part I," *IEEE Robotics & Automation Magazine*, vol. 13, no. 2, pp. 99–110, 2006.